Practical, Hands-on Tips for VB.NET and C# !

1st EDITION

SUPPLEMENT TO

www.vbpj.com • www.vcdj.com

VisualStudio™ Magazine

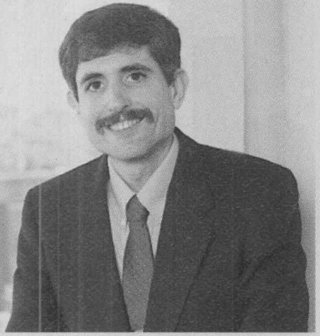
A FAWCETTE TECHNICAL PUBLICATION, SEPTEMBER 2001 VOL. 11, NO. 10

101 TECH TIPS

Includes Tips For:

- VB5 and VB6
- VB.NET
- C#
- SQL Server
- ASP & IIS
- XML

For Visual Studio Developers



The .NET Learning Curve

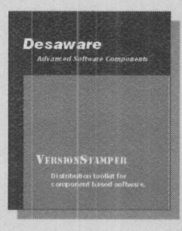
Dan Appleman, leading author on VB

The biggest challenge .NET poses to Windows developers today is not deployment - it's education. The .NET Framework is so huge and so different, that you'll need to invest significant time studying and experimenting with .NET before you start deploying real applications. How does one go about learning .NET? Expensive courses and conferences? Books? Articles? Regardless, it is essential to first learn the core concepts of .NET. That is the foundation on which you can base further learning, and that's the focus of my new book **"Moving to VB.NET: Strategies, Concepts and Code"**. You can read the introduction and sample text at www.desaware.com. There you'll also read about SpyWorks 6.3 with .NET support, VersionStamper's support for .NET applications that use COM components, and new products designed specifically for .NET. ..... Dan

NT Service Toolkit
Full featured services with VB6!

Visual Basic programmers have long searched for a good way to create NT Services using Visual Basic. Yet every solution they found had serious limitations and compromises - until now. Desaware's NT Service Toolkit provides all the power of a C++ service from VB - but the services are far easier to create and debug!

- **Debug your service using the VB environment** - while it is running as a service!
- Automatic **background threads** for waiting on NT synchronization objects.
- Access and control services through **COM or DCOM**.
- Expose client objects via COM or DCOM.
- Client objects run on a **thread pool for scalability**.
- Create background threads for asynchronous operations.
- Create control panel applets to manage your services.
- All service notifications (including those new to Win 2000).
- Simulator allows testing without installing as a service.
- Impersonation allows acting on behalf of clients.
- Access to all service configuration parameters.

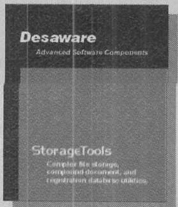


Escape DLL Hell!

VersionStamper helps eliminate incompatibility problems that can occur when distributing component based applications - **including .NET applications that use COM components**. Your application can analyze a client system using an embedded dependency list or data from your Web or FTP site. Problems can be resolved by notifying the user, reporting via Email, or **automatic download** of the correct components from your Web or FTP site.

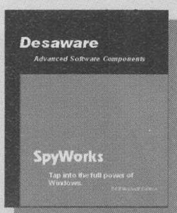
New! Event Log Toolkit

Create custom event sources with this easy to use toolkit. Eliminates the problem with VB6 in which all events are logged as coming from VBRuntime. Event sources are easy to distribute, self-installing, and support multiple languages. Included code demonstrates advanced event reporting API calls.



Data Storage for VB, VBA and Internet Applications

StorageTools allows you to create complex documents, each of which can be subdivided into multiple named blocks of data. StorageTools actually gives you **more data storage flexibility than most databases, without the overhead!**

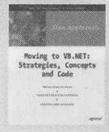


Desaware
Advanced Software Components
SpyWorks
Tap into the full power of Windows

SpyWorks® 6.3 - The ultimate low level toolkit, now compatible with VB.NET and C# for VS.NET Beta 2

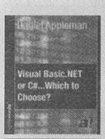
Advanced Subclassing and Hooks - In-and cross process subclassing for VB6 and .NET! Use **Windows Hooks** to intercept messages and keys for a window, process or the entire system. **Create Background threads** for your VB6 DLL COM objects. Create true **Function Exports** from your VB6 DLLs. Handle Internet/Intranet communications with our **Winsock Component** - with VB source. Create **Control Panel Applets. Implement or call ANY interface** - No type library. *And much more....*

Books



Moving to VB.Net: Strategies, Concepts and Code

VB.Net is not Visual Basic. Porting is stupid. COM is "dead". These are just a few of the things you'll learn as Dan takes you on a journey unlike any other into the world of VB.Net. Covers adoption strategies, unlearning VB6 concepts that are fatal in VB.Net, and analysis of language changes that goes beyond the documentation. APRESS, ISBN 1-893115-97-6



Visual Basic.NET or C# - Which to Choose? - A new PDF based e-book comparing VB.NET and C# available now at www.desaware.com/VBorCsharp.htm.

How Computer Programming Works -The ideal book for beginning programmers. Teaches the fundamentals of programming using illustrations and real world analogies. APRESS, ISBN 1-893115-23-2

Also available

Dan Appleman's Visual Basic Programmer's Guide to the Win32 API - The core Windows Software Development Kit rewritten for VB programmers. **Dan Appleman's Win32 API Puzzle Book and Tutorial for VB Programmers** teaches you to call any API function from VB5 & 6. **Developing COM/ActiveX Components with Visual Basic 6.0.** The essentials of COM and component development for VB.

For more information, please visit...

www.desaware.com

Desaware Inc.
1100 E. Hamilton Ave., Suite 4
Campbell, CA 95008
tel: (408) 377-4770 fax: (408) 371-3530
email: support@desaware.com

Welcome to the First Edition of the *VSM* Technical Tips Supplement!

The editors of *Visual Studio Magazine* are pleased to bring you these invaluable tips, techniques, and workarounds, submitted and reviewed by professional developers. Instead of typing the code published here, download the tips for free from the *VSM* Web site at www.vbpj.com or www.vcdj.com.

We know you've uncovered your own tips and tricks—send them to us at vsmtips@fawcette.com. Include a clear explanation of what the technique does, why it's useful, and what language(s) and version(s) it applies to. Please limit code length to 20 lines. Don't forget to include your mailing address, and let us know your compensation preference per published tip: $25, a new one-year *VSM* subscription, or a one-year extension to your existing *VSM* subscription.

VB.NET
Level: Intermediate

Return Strings From an API

In .NET, strings are immutable: When you pass them out to an API, you can't modify them. However, VB.NET applies the VBByRefStr unmanaged type-marshaling attribute to the string. This allows VB.NET to create a temporary buffer, copy that back to a new string, then point the original string to the new string:

```
Public Declare Function GetWindowText _
   Lib "User32.Dll" _
   (ByVal hwnd As Int32, _
   ByVal lpString As String, _
   ByVal cch As Int32) As Int32
```

To use this declaration, simply initialize the string to the right size:

```
Dim s As String = Space(256)
Dim rtn as Int32 = GetWindowText(hwnd, s, 256)
```

The API declaration is equivalent to:

```
Public Declare Function GetWindowText _
   Lib "User32.Dll" (ByVal hwnd As Int32, _
   <System.Runtime.InteropServices.MarshalAs( _
   Runtime.InteropServices.UnmanagedType.VBByRefStr)> _
   ByRef lpString As String,
   ByVal cch As Int32) As Int32
```

VB.NET, however, doesn't allow you to specify that marshaling attribute on parameters, so you must use the first declaration. Use a StringBuilder object as an alternative to using the VBByRefStr attribute.

—Bill McCarthy, Barongarook, Victoria, Australia

VB4, VB5, VB6
Level: Beginning

Count the Number of Elements in an Array

This function computes the number of elements of any one-dimensional array—it sure beats ripping open the SAFEARRAY array descriptor. Use it when you're not sure whether an array is one- or zero-based:

```
Public Function CountElements( _
   ByVal SimpleArray As Variant) As Long
   ' Ignore error if array not dimensioned
   On Error Resume Next
   If Not IsArray(SimpleArray) Then Exit Function
   CountElements = Abs((LBound(SimpleArray)) - _
      (UBound(SimpleArray))) + 1
End Function
```

—Monte Hansen, Ripon, Calif.

VB4/32, VB5, VB6
Level: Beginning

Extend Registry Functionality

An (undocumented) feature of VB's native *Setting Registry functions is that they can create and access multilayer hierarchies such as this:

```
VB and VBA Project Settings
Application
      Plugin
         Section
            Subsection
               Key = "Value"
```

You can do this easily—simply add a "\" character between the parent entry and its child entry. Then you can use the Registry as you'd use a folder with subfolders. Check out these code examples that create and read structures such as the preceding one:

```
Call SaveSetting("Application\Plugin", _
   "Section\Subsection", "Key", "Value")
.Print SaveSetting("Application\Plugin\Section", _
   " Subsection", "Key", "Value")
```

—Chris Hynes, Fort Washington, Md.

C#
Level: Beginning

Use String Literals to Simplify Paths

When you need to set a string to a local or network path, use a string literal to avoid writing repeating backslashes. For instance, this code:

```
string sLocalPath = "C:\\directory\\file.txt";
string sNetworkPath =
"\\\\machinename\\directory\\file.txt";
```

Becomes this:

```
string sLocalPath = @"C:\directory\file.txt";
string sNetworkPath =
@"\\machinename\directory\file.txt";
```

—Robert Lair, Springboro, Ohio

VB5, VB6
Level: Beginning

Let VB Spell the Control Name for You

I love the way VB's IntelliSense drops down a list of control names, properties, and methods as soon as I type the period after the object's name. But when you're coding in a form's own module and referencing the control name directly, without an object qualifier, you can still get that cool dropdown list. First type "Me." (with the period) and pick the name. Feel free to delete the "Me." qualifier later. Here's an example:

```
Call clsDatabase.LoadMyListBox(Me.MylistBox)
```

—Frank Ramage, Laurel, Md.

VB4/32, VB5, VB6, VBS, SQL 7.0 and up
Level: Intermediate

Preprocess Nulls in Your Recordset

I've seen several examples of VB code dealing with nulls in recordsets before the value is assigned to a VB control. I often use the Transact-SQL IsNull statement when working with SQL Server (version 7.0 and later) queries. SQL Server uses IsNull to deal with the null value if it arises so I don't have to write additional code to handle nulls when I process a recordset. For example, in this code that reads CustomerID and EmailAddress from a Customers Table, SQL Server returns the value "na" if the EmailAddress field is null:

```
set rsCustomerDetails = cn.Execute("Select " & _
   "CustomerID, IsNull(EmailAddress,'na') as " & _
   "EmailAddress from Customers")
```

You can't update the recordset because IsNull appears in the Select statement. But you'll find many circumstances where this doesn't matter, such as when using VBScript to build a table in a Web page.

—Robert Bryan, Downer, Australian Capital Territory, Australia

VB5, VB6
Level: Intermediate

Add a Picture Preview Property Page

Defining a public property in a user control as Picture (or StdPicture) provides the standard ellipsis next to the property name in VB's property viewer automatically. This pops up a standard dialog to load an image control.

Let's say you have this code in a user control:

```
Public Property Get Picture() As Picture
   Set Picture = UserControl.Picture
End Property

Public Property Set Picture( _
   ByVal newPicture As Picture)
   Set UserControl.Picture = newPicture
   PropertyChanged "Picture"
End Property
```

You can add a standard VB property page to the control that provides the standard preview window so you can see what you're loading. To do this, you open the UserControl in design mode and select the PropertyPages property. You'll see a dialog with three or four choices: StandardPicture, StandardFont, StandardColor, and (for VB6) StandardDataFormat. Simply check the ones you wish to have a custom property added to the UserControl's other properties. Note: Just because you add the property pages doesn't mean you can access them immediately. You need to assign the page to specific properties using the Procedure Attributes dialog.

—John Cullen, Pedroucos, Portugal

VB6
Level: Intermediate

Split Strings Cleanly, Redux

In the 10th Edition of the "101 Tech Tips for VB Developers" supplement [*Visual Basic Programmer's Journal* February 2000], the "Split Strings Cleanly" function splits an array containing more than one delimiter in a row efficiently. This functions works great for one delimiter, but what if you want to split an array on more than one delimiter? Adding a few lines of code and using recursion can enhance the function to handle multiple delimiters.

When more than one delimiter is passed into the function, you rejoin the filtered array using the next delimiter, drop the current delimiter from the delimiter list, and call the function again:

```
Public Function CleanSplit2( _
   ByVal Expression As String, _
   Optional ByVal Delimiters As String = " ", _
   Optional ByVal Limit As Long = -1, _
   Optional Compare As VbCompareMethod = _
   vbBinaryCompare) As Variant

   Dim Substrings() As String
   Dim OneDelimiter As String
   Dim I As Long

   OneDelimiter = Mid$(Delimiters, 1, 1)
   Substrings = Split(Expression, OneDelimiter, _
      Limit, Compare)
   For I = LBound(Substrings) To UBound(Substrings)
      If Len(Substrings(I)) = 0 Then
         Substrings(I) = OneDelimiter
      End If
   Next I
   If Len(Delimiters) = 1 Then
      CleanSplit2 = Filter( _
         Substrings, OneDelimiter, False)
   Else
      CleanSplit2 = _
         CleanSplit2(Join( _
         Filter(Substrings, OneDelimiter, False), _
         Mid$(Delimiters, 2, 1)),
         Mid$(Delimiters, 2), Limit, Compare)
   End If
End Function
```

—Stephen Sayabalian, Waltham, Mass.

VB3, VB4, VB5, VB6
Level: Intermediate

Keep Your Projects Intact

VB has always gone out of its way to take care of mundane housekeeping tasks without bothering you with the details. But sometimes the best intentions can create unintended problems. When you work with a project, VB automatically keeps track of the project's files by maintaining their entries in the VBP project file (MAK in VB3). When you move files around or bring in files from other projects, VB edits the path information for those files, sometimes creating an incomprehensible mess of upward-moving relative paths littered with "\..\" steps. The result can be catastrophic—you can inadvertently edit a different project's source code, or you can end up missing files when you move a project to another directory or computer. To avoid these problems, first make sure all your working files are in the directories you intended, then edit the VBP file manually in Notepad to remove any visually ambiguous path descriptions. Make sure you exit the VB IDE before editing the VBP file.

—Ron Schwarz, Hart, Mich.

VB3, VB4, VB5, VB6
Level: Beginning

A New Look at the Select Case

Who says the Select Case can evaluate only one statement? Try Select Case True instead of the If...ElseIf...End If blocks. Select Case's more eloquent style makes code easier to read, write, and maintain. Consider both these routines that reset the controls on a form:

```
' If...ElseIf...End If Version
Public Sub IfResetForm(frmForm As Form)
   Dim c As Control

   For Each c In frmForm.Controls
      If TypeOf c Is TextBox Then
         c.Text = ""
      ElseIf TypeOf c Is ComboBox Then
         c.ListIndex = -1
         If c.Style <> vbComboDropdownList Then
            c.Text = ""
         End If
      ElseIf TypeOf c Is ListBox Then
         c.ListIndex = -1
      ElseIf TypeOf c Is CheckBox Then
         c.Value = vbUnchecked
      ElseIf TypeOf c Is OptionButton Then
         c.Value = False
      End If
   Next c
End Sub

' Select Case True Version
Public Sub SelResetForm(frmForm As Form)
   Dim c As Control

   For Each c In frmForm.Controls
      Select Case True
         Case TypeOf c Is TextBox
            c.Text = ""
         Case TypeOf c Is ComboBox
            c.ListIndex = -1
            If c.Style <> vbComboDropdownList Then
               c.Text = ""
            End If
         Case TypeOf c Is ListBox
            c.ListIndex = -1
         Case TypeOf c Is CheckBox
            c.Value = vbUnchecked
         Case TypeOf c Is OptionButton
            c.Value = False
      End Select
   Next c
End Sub
```

The Select Case True routine is easy to read and debug, whereas the If...ElseIf...End If routine gives the impression of nesting and might be more difficult to evaluate at a glance. The Select Case True executes the code block of the first true statement it encounters; however, code tends to flow better when you construct a list of Case statements rather than a series of ElseIf blocks.

—Michael C. Stahr, Oxford, Ohio

VB4/32, VB5, VB6, SQL Server 6.5 and up, Oracle 8*i* and up
Level: Intermediate

Change Oracle and SQL Server Passwords

You can change database passwords from within VB to control more of your application's security and limit your dependence on an external DBA. This function updates a database password for either Oracle or SQL Server:

```
Function UpdateLogin(pbOracle As Boolean, _
   padoConn as ADODB.Connection, _
   pstrUserId As String, _
   pstrCurPassword As String, _
   pstrNewPassword As String) As Boolean

   Dim strSQL As String
   On Error GoTo ErrHandler
   UpdateLogin = True
   If (pbOracle) Then
      strSQL = "ALTER USER " & pstrUserId & _
         " IDENTIFIED BY " & pstrNewPassword
   Else
      strSQL = "sp_password '" & _
         pstrCurPassword & "', '" & _
         pstrNewPassword & "'"
   End If
   padoConn.Execute strSQL
   Exit Function
ErrHandler:
   UpdateLogin = False
   Exit Function
End Function
```

To use this, you should connect to the database using the account you're changing.

—Andy Clark, Richmond, Va.

VB4/32, VB5, VB6, VBA, VBS
Level: Intermediate

Generate OLE DB Connection Strings

Many VB projects need a database connection string. But there's no easy way to generate an OLE DB connection string without adding a DataEnvironment to your project, setting the values on the Data Link Properties dialog by selecting Properties from the Connection1 object context menu, and finally retrieving the value in the ConnectionSource property as your connection string.

For a better way, simply paste these nine lines of code into a text file with a VBS (VBScript) extension, and double-click on the file:

```
Dim oDataLinks, sRetVal
Set oDataLinks = CreateObject("DataLinks")
On Error Resume Next ' Trap Cancel button
sRetVal = oDataLinks.PromptNew
On Error Goto 0
If Not IsEmpty(sRetVal) Then ' Didn't click Cancel
InputBox "Your Connection String is listed below.", _
   "OLEDB Connection String", sRetVal
End If
Set oDataLinks = Nothing
```

Follow the usual prompts to place the resulting connection string in an input box for easy cut-and-pasting. Now any time you need a connection string for an OLE DB data source, it's only a double-click away.

Note: If you're using VB, you can add a reference to the Microsoft OLE DB Service Component 1.0 Type Library (OLEDB32.dll) and use the Object Browser to explore the additional interfaces the DataLinks object exposes.

—Anthony T. Petro, Centennial, Colo.

VB6
Level: Advanced

Continue After Hitting an Error

If you use VB6 to write COM programs that raise errors, it seems impossible to continue after hitting one of them. However, the (almost undocumented) commands ALT+F8 and ALT+F5 let you step and run past an error, respectively, into the error-handling code or—more importantly—into the code that called the procedure where the error occurred (such as a C++ client). This can be a real lifesaver if you're coding in both C++ and VB and the programs have to play nicely. You can directly receive error results raised in the VB code instead of hacking around it by setting breakpoints in the client, setting the next line of execution to something that returns to the caller, and using the debugger to save the error code into a variable on the client. Why the "run," "step over," and similar buttons don't do what ALT+F8 and ALT+F5 do is beyond me.

—Michael Nyman, Tualatin, Ore.

VB5, VB6
Level: Beginning

Generate Rule-Based Random Strings
Use this function to generate random strings that abide by certain criteria. It's perfect for password generators or strings used in a challenge/response authentication scheme:

```
Public Enum RandomStringOptions
    rsoAllChars = 0
    rsoAllCharsExtended = 1
    rsoKeyboardChars = 2
    rsoAlphaNumericChars = 3
End Enum

Public Function RandomString(Optional ByVal _
   MinLength As Long = 20, _
   Optional ByVal MaxLength As Long = 29, _
   Optional ByVal ExclusionCharacters As String = _
   " ", Optional ByVal RandomOption As _
   RandomStringOptions = rsoAlphaNumericChars) _
   As String
'==================================================
' Generates a random string using ...
' Max/MinLength: Determines the minimum and
' maximum size of the string.
' ExclusionCharacters: Characters that cannot
' appear in the random string.
' RandomOption: Special options used to define
' additional rules.
'==================================================
   ' Where random string is built
   Dim Buffer()    As Byte
   ' Next character to test
   Dim NextChar    As Byte
   ' The lower range of the char table
   Dim iCharLo     As Integer
   ' The upper range of the char table
   Dim iCharHi     As Integer
   Dim i           As Long

   ' Sanity check
   If MinLength < 1 Or MaxLength < MinLength Then
      Err.Raise 5, App.ProductName
   End If

   If RandomOption = rsoKeyboardChars Then
      ' -- only keyboard characters are supported
      ' Characters 32 through 126 are keyboard
      ' characters
      iCharLo = 32: iCharHi = 126
   ElseIf RandomOption = rsoAlphaNumericChars Then
      ' This range included entire alphanumeric
      ' characters
      iCharLo = 48: iCharHi = 122
   ElseIf RandomOption = rsoAllCharsExtended Then
      ' -- we can use the entire "standard" ascii
      ' character set
      iCharLo = 0: iCharHi = 127
   Else ' RandomOption = rsoAllChars
      ' we can use the entire character set,
      ' including extended characters
      iCharLo = 0: iCharHi = 255
   End If

   ' Fire up the random number generator
   Randomize Timer
   ' Size the buffer to fit a random number size
   ' within the desired string length range.
   ReDim Buffer(1 To Int((MaxLength - MinLength _
      + 1) * Rnd + MinLength))

   ' Loop through the output buffer
   For i = LBound(Buffer) To UBound(Buffer)
      ' Loop until "good" character is selected
      Do
         ' Get a random character in the character
         ' set range
         NextChar = Int((iCharHi - iCharLo + 1) * _
            Rnd + iCharLo)

         ' Make sure not in exclusion list
         If InStr(ExclusionCharacters, _
            Chr(NextChar)) = 0 Then
            ' Check if AlphaNumeric?
            If RandomOption = rsoAlphaNumericChars _
               Then
               Select Case NextChar
               Case 48 To 57, 65 To 90, 97 To 122
                  ' within the alphanumeric range
                  ' of characters
                  Exit Do
               Case Else
                  ' just keep on looping until
                  ' alphanumeric
                  ' character generated.
               End Select
            Else
               ' we have a non-excluded char
               Exit Do
            End If
         End If
      Loop
      ' Assign this char, and get next
      Buffer(i) = NextChar
   Next i

   ' Return the resulting string
   RandomString = StrConv(Buffer, vbUnicode)
End Function
```

—Monte Hansen, Ripon, Calif.

C#
Level: Beginning

Close a Windows Form
A Close button, which closes a form when the user clicks on it, is one of the most common interface controls added to a Windows form. Unfortunately, the wizard does not generate the code for you, so you must do it manually. Add a button to the form; set its text to Close, Cancel, or Exit; and give it a meaningful name such as m_CloseButton. Next, create a Click event method handler (such as OnCloseButtonClick) and add a new delegate, initialized with that handler to the Close button's Click event:

```
public class MyForm : Form
{
   protected Button m_CloseButton;
   public MyForm()
   {
      InitializeComponent();
      CancelButton = m_CloseButton;
   private void InitializeComponent()
   {
      m_CloseButton = new Button();
      m_CloseButton.Click +=
         new EventHandler(OnCloseButtonClick);
      Controls.Add(m_CloseButton);
   }
   protected void OnCloseButtonClick(object sender,
EventArgs e)
   {
      Close(); //calls Dispose() for you as well
   }
}
```

You can also double-click on the Close button in the visual designer and let Visual Studio.NET generate this code for you. In your Close button Click event handler, simply call your base class (System.Windows.Forms.Form) Close() method to close the form. In addition to closing the form, the Form.Close() implementation also calls Dispose() for you, so you don't need to call it explicitly yourself.

Finally, you need to handle the event of the user hitting the Escape key. The convention in Windows is that this action should close the form, just as if the user clicked on the Close button. In your form constructor, after the call to InitializeComponent(), set your base class CancelButton property to the Close button you just added. This will redirect the Escape event to your button, as if the user clicked on it.

—Juval Lowy, San Jose, Calif.
author of *COM+ Services - Mastering COM and .NET Component Services* [O'Reilly, 2001]

VB6, SQL Server 6.5 and up
Level: Intermediate

Let MTS Handle Transaction Management

When you call a stored procedure in SQL Server that performs data manipulation on the database from a Microsoft Transaction Server (MTS) transaction, let MTS handle all the transaction management. Don't put a BEGIN TRANSACTION | COMMIT TRANSACTION | ROLLBACK TRANSACTION in the stored procedure. The transaction you create in the stored procedure doesn't enlist in the MTS transaction, so MTS isn't notified when you handle the SQL errors manually. This means an error in your stored procedure won't force the rollback of the MTS transaction's other parts. The MTS transaction returns a success notification even when part of the transaction failed.

—Jason Rein, Thompson's Station, Tenn.

VB4/32, VB5, VB6
Level: Beginning

Return File Version Info

Regarding the "Retrieve File Version Information" tip in the 11th Edition of the "101 Tech Tips for VB Developers" supplement [*Visual Basic Programmer's Journal* March 2001], I have a shorter function that achieves the same task. To use the FileSystemObject, you need to reference the Microsoft Scripting Runtime:

```
Public Function GetExecutableFileVersion(ByVal _
   Filename As String) As String
   Dim FileObj As Scripting.FileSystemObject

   ' Create Object
   Set FileObj = New Scripting.FileSystemObject

   If FileObj.FileExists(Filename) Then
      GetExecutableFileVersion = _
         FileObj.GetFileVersion(Filename)
   End If

   ' Free Object
   Set FileObj = Nothing
End Function
```

—Simon Murrell, Bedfordview, Gauteng, South Africa

VB4/32, VB5, VB6, VBS
Level: Intermediate

☆☆☆☆☆ Five Star Tip

Create ISAM Files Out of Thin Air

Visual Basic Programmer's Journal once published a tip on how to use undocumented Jet/SQL features to create a new ISAM file in the format of your choice—such as Excel, dBase, Paradox, HTML, and Lotus—without having to use automation with the object model or even having the destination file type's application on the user's machine ["Export Data to ISAM Databases," 6th Edition of the "101 Tech Tips for VB Developers" supplement, *Visual Basic Programmer's Journal* February 1998].

Microsoft still says that method doesn't exist. Here's another it says doesn't exist, but it does as of ADO/ADOX 2.1. Start a new VB project and add a reference to ADO and ADOX (Microsoft ADO Extensions for DDL and Security):

```
Dim cn As Connection
Dim cat As Catalog
Dim tbl As Table
Dim fld As Column
Set cn = New Connection
With cn
   .ConnectionString = _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Extended Properties=Excel 8.0;Data Source=" _
      & App.Path & "\anewfile.xls"
   .Open
End With

Set cat = New Catalog
With cat
   .ActiveConnection = cn
   Set tbl = New Table
   tbl.Name = "ANewSheet"
   Set fld = New Column
   fld.Name = "MyCol1"
   fld.Type = adWChar
   fld.DefinedSize = 30
   tbl.Columns.Append fld
   .Tables.Append tbl

   Set tbl = New Table
   tbl.Name = "Another"
   Set fld = New Column
   fld.Name = "Wubba"
   fld.Type = adWChar
   fld.DefinedSize = 10
   tbl.Columns.Append fld
   .Tables.Append tbl

   cat.Tables.Refresh
End With
Set fld = Nothing
Set tbl = Nothing
Set cat = Nothing
cn.Close
Set cn = Nothing
```

Run it to see your Excel file created with a Worksheet named NewSheet and a column named/typed as you specified.

Use this to wow your users with multisheet reports. Once you create the sheets, you can use ADO to connect to the files and manipulate them as you would any other ADO data source. But the coolness of this technique lies not in the fact that you can manipulate an ISAM data source once you have one on your machine—that has always been easy. The coolness is the ability to create the files out of thin air in the first place using only ADO.

Like the original SQL method, this approach creates most other ISAMs too. Simply replace the *Extended Properties=* value with the specifier you desire, such as *Extended Properties=dBase IV;*, *Extended Properties=Paradox 4.x;*, and so on.

—Robert Smith, Kirkland, Wash.

VB3, VB4, VB5, VB6
Level: Beginning

Tweak the Key and Mouse Events

Many VB objects have KeyDown, KeyUp, MouseDown, MouseUp, and MouseMove events. They're unique in one respect: They have built-in limitations on what they're able to do. You often need to coordinate these events with things outside their immediate scope to use them effectively. For example, you might need your app to perform a repetitive action while a mouse button is down.

The MouseDown event fires only once—when the button is clicked—so you can't determine the mouse state *outside* the MouseDown event easily. Handle these situations by using a variable with sufficient scope to address the required code. For example, if you're coding all the mouse-oriented functionality within the form that contains the control in question, declare a form-level Boolean variable and use it to track the mouse's state. Set the variable to True when the MouseDown event fires, and to False when the MouseUp event fires. When your code executes, have it check the mouse state by testing the variable's current value. The Mouse events then maintain the variable's state automatically, and you can track the variables', hence the mouse, state from outside the mouse events.

—Ron Schwarz, Hart, Mich.

VB5, VB6
Level: Advanced

Get Dynamic Array Information

Use the GetSafeArrayInfo function to rip the lid off a SAFEARRAY. It allows the caller to identify the number of dimensions and number of elements for each dimension (among other things). Element information for each dimension is stored in a one-based subarray of SAFEARRAYBOUND structures (rgsabound):

```
Public Type SAFEARRAYBOUND
    ' # of elements in the array dimension
    cElements As Long
    ' lower bounds of the array dimension
    lLbound As Long
End Type
Public Type SAFEARRAY
   ' Count of dimensions in this array.
   cDims As Integer
   ' Flags used by the SafeArray
   ' routines documented below.
   fFeatures As Integer
   ' Size of an element of the array.
   ' Does not include size of
   ' pointed-to data.
   cbElements As Long
   ' Number of times the array has been
   ' locked without corresponding unlock.
   cLocks As Long
   ' Pointer to the data.
   ' Should be sized to cDims:
   pvData As Long
   ' One bound for each dimension.
   rgsabound() As SAFEARRAYBOUND
End Type

Private Declare Sub CopyMemory Lib "kernel32" Alias _
   "RtlMoveMemory" (ByVal lpDest As Long, ByVal _
   lpSource As Long, ByVal nBytes As Long)

Public Function GetSafeArrayInfo(TheArray As _
   Variant, ArrayInfo As SAFEARRAY) As Boolean
'==================================================
' Fills a SAFEARRAY structure for the array.
' TheArray: The array to get information on.
' ArrayInfo: The output SAFEARRAY structure.
' RETURNS: True if the array is instantiated.
'==================================================
   ' Pointer to the variants data item
   Dim lpData       As Long
   ' the VARTYPE member of the VARIANT structure
   Dim VType        As Integer
   Const VT_BYREF   As Long = &H4000&

   ' Exit if no array supplied
   If Not IsArray(TheArray) Then Exit Function

   With ArrayInfo
      ' Get the VARTYPE value from the first 2 bytes
      ' of the VARIANT structure
      CopyMemory ByVal VarPtr(VType), ByVal _
         VarPtr(TheArray), 2

      ' Get the pointer to the array descriptor
      ' (SAFEARRAY structure)
      ' NOTE: A Variant's descriptor, padding &
      ' union take up 8 bytes.
      CopyMemory ByVal VarPtr(lpData), ByVal _
         (VarPtr(TheArray) + 8), 4

      ' Test if lpData is a pointer or a pointer to
      ' pointer.
      If (VType And VT_BYREF) <> 0 Then
         ' Get real pointer to the array descriptor
         ' (SAFEARRAY structure)
         CopyMemory ByVal VarPtr(lpData), ByVal _
            lpData, 4
         ' This will be zero if array not
         ' dimensioned yet
         If lpData = 0 Then Exit Function
      End If

      ' Fill the SAFEARRAY structure with the array
      ' info
      ' NOTE: The fixed part of the SAFEARRAY
      ' structure is 16 bytes.
      CopyMemory ByVal VarPtr(ArrayInfo.cDims), _
         ByVal lpData, 16

      ' Ensure the array has been dimensioned before
      ' getting SAFEARRAYBOUND information
      If ArrayInfo.cDims > 0 Then
         ' Size the array to fit the # of bounds
         ReDim .rgsabound(1 To .cDims)

         ' Fill the SAFEARRAYBOUND structure with
         ' the array info
         CopyMemory ByVal VarPtr(.rgsabound(1)), _
            ByVal lpData + 16, _
            ArrayInfo.cDims * Len(.rgsabound(1))

         ' So caller knows there is information
         ' available for the array in output
         ' SAFEARRAY
         GetSafeArrayInfo = True
      End If
   End With
End Function
```

—Monte Hansen, Ripon, Calif.

For even more tricks and tips go to
www.vbpj.com or www.vcdj.com

VB4/32, VB5, VB6, SQL Server 6.5 and up, Oracle 8*i* and up
Level: Intermediate

Compare Oracle and SQL Server Strings

Oracle and SQL Server treat strings slightly differently—they don't trim blank and null characters identically. So you'll have problems comparing string data between Oracle and SQL Server. The solution—trim both blanks and nulls from all strings:

```
Private Function MatchStrings(adoFldOra As _
   ADODB.Field, adoFldSQLServ As _
   ADODB.Field) As Boolean
Dim strOracle As String
Dim strSQLServ As String
   strOracle = TrimNulls(adoFldOra.Value)
   strSQLServ = TrimNulls(adoFldSQLServ.Value)
   MatchStrings = (strOracle = strSQLServ)
End Function

Private Function TrimNulls(pstrIn As String) As _
   String
Dim ndx As Integer
Dim pos As Integer
Dim strWork As String
   strWork = ""
   pos = 0
   ndx = 1
   Do While ((pos = 0) And (ndx <= Len(pstrIn)))
      If (Asc(Mid(pstrIn, ndx, 1)) <> 0) _
         And (Mid(pstrIn, ndx, 1) <> " ") Then
         pos = ndx
      End If
      ndx = ndx + 1
   Loop
   If (pos = 0) Then
      TrimNulls = ""
      Exit Function
   End If
   strWork = Mid(pstrIn, pos)
   ndx = Len(strWork)
   pos = 0
   Do While ((pos = 0) And (ndx > 0))
      If (Asc(Mid(pstrIn, ndx, 1) <> 0) And _
         (Mid(pstrIn, ndx, 1) <> " ")) Then
         pos = ndx
      End If
      ndx = ndx - 1
   Loop
   If (pos = 0) Then
      TrimNulls = ""
      Exit Function
   End If
   TrimNulls = Left(strWork, pos)
End Function
```

—Andy Clark, Richmond, Va.

VB.NET
Level: Beginning

Clarify Procedure Attributes With Line Continuation

Add readability to your attribute assignments by placing them on their own line with an underscore line-continuation character:

```
<Obsolete("Use NewCalc instead",True)> _
   Public Sub Calc()
   ...
End Sub
```

—Jonathan Goodyear, Orlando, Fla.

VB.NET, C#
Level: Beginning

Reading Console Output When Working in the IDE

When you run a console from the IDE, the console often disappears before you get the chance to view the output. You can work around this by using Console.Read() to pause the program until you hit the Enter key.

VB.NET:

```
   Console.WriteLine("Press Enter to close this window")
   Console.Read()
```

C#:

```
   Console.WriteLine("Press Enter to close this window");
   Console.Read()
```

You can also change the build output type to Windows Application. When you change a console application's output type to Windows Application, the console's output gets redirected to the IDE's output window where you can view the output after the application has finished running as well as while the application is running.

—Jonathan Goodyear, Orlando, Fla.

SQL Server 6.5 and up
Level: Beginning

Find the Cause of Query Malfunctions

When debugging a SELECT query, add an absolute true condition as the first condition of the WHERE clause:

```
SELECT
   au_lname,
   au_fname
FROM
   authors
WHERE
   1=1 --absolute true condition
   and state = 'CA'
   and contract = 1
```

That way, you can comment out one or more of the real conditions in the WHERE clause using the "--" comment character sequence to narrow down which condition(s) cause the query to malfunction. When you're done debugging your query, remove the absolute true condition as well as the "and" in front of the first real condition.

—Jonathan Goodyear, Orlando, Fla.

VB.NET
Level: Beginning

Know the Differences Between CStr and .ToString

VB.NET's CStr() method is locale-aware, which means it uses the locale at run time to determine how to format the string. The .ToString method is locale-neutral and is generally quicker. If you need to format strings according to the end user's regional settings, use CStr. Otherwise, use .ToString, which works on all objects. CStr works only on objects that implement IFormattable.

Note: Neither works exactly like VB5/6 CStr, which returns a localized string based on compile-time settings.

—Bill McCarthy, Barongarook, Victoria, Australia

VBS, ASP/IIS 4.0 and up
Level: Intermediate

Upload Files to Active Server Pages
Many companies sell ActiveX objects for uploading files to Active Server Pages. However, you can easily write a bit of VBScript to handle the uploads yourself. The Request object has all the uploaded data:

```
<!--
   HTML for the upload form
   'uploadform.html'
-->
<html><body>
<form action="upload.asp" method="post"
enctype="multipart/form-data">
<input type=file name="file1" size=20><br>
<input type=file name="file2" size=20><br>
<input type=submit>
</body></html>

<!--
   ASP Code for the upload
   'upload.asp'
-->
<html><body>
<%
   ' This code is needed to "initialize" the
   ' retrieved data
   Dim q
   q = Chr(34)
   ' All data
   Dim aAllDataB, aAllData, x, aHdr
   aAllDataB = Request.BinaryRead(Request.TotalBytes)
   ' It comes in as unicode, so convert it to ascii
   For x = 1 To LenB(aAllDataB)
      aAllData = aAllData & Chr(AscB(MidB( _
         aAllDataB, x, 1)))
   Next
   ' The "header" is a unique string generated by the
   ' system to indicate the beginning
   ' and end of file data
   aHdr = Left(aAllData, Instr(aAllData,vbCrLf)+1)
%>

<%
   ' Here's where your code goes.
   ' In this example, "file1" and "file2" are the
   ' field names specified within the form of the
   ' upload submission page.
   Response.Write "file1: Filename = " & _
      GetFilename("file1") & "<br>"
   Response.Write GetFileData("file1") & "<br><br>"

   Response.Write "file2: Filename = " & _
      GetFilename("file2") & "<br>"
   Response.Write GetFileData("file2") & "<br><br>"

   ' Writing out the file data like this only looks
   ' OK when the uploaded file is some kind of text
   ' - images and things like that probably just need
   ' to be saved or otherwise acted upon.
   Response.Write Replace(aAllData,vbCrLf,"<br>")

   Dim aFilename
   ' aFilename equates to the original filename,
   ' except saved in the root path of the server.
   ' The root path must have Change rights for the
   ' default Internet user.
   aFilename = Server.MapPath("\") & "\" & _
      GetFileName("file1")
   Call SaveFile("file1", aFilename)
   aFilename = Server.MapPath("\") & "\" & _
      GetFileName("file2")
   Call SaveFile("file2", aFilename)
%>
</body></html>
<%
   ' These are functions used to retrieve the data
Function GetFileName(aField)
   Dim x2, i
   x = Instr(aAllData, aHdr & _
      "Content-Disposition: form-data; name=" & _
      q & aField & q)
   x = Instr(x, aAllData, "filename=" & q)
   x2 = Instr(x, aAllData, vbCrLf)
   For i = x2 To x Step -1
      If Mid(aAllData,i,1) = "\" Then
         x = i - 9
         Exit For
      End If
   Next
   GetFileName = Mid(aAllData, x+10, x2-(x+11))
End Function
Function GetFileData(aField)
   Dim x2
   x = Instr(aAllData, aHdr & _
      "Content-Disposition: form-data; name=" & _
      q & aField & q)
   x = Instr(x, aAllData, vbCrLf)
   x = Instr(x+1, aAllData, vbCrLf)
   x = Instr(x+1, aAllData, vbCrLf) + 2
   x2 = Instr(x, aAllData, Left(aHdr,Len(aHdr)-2))
   GetFileData = Mid(aAllData, x+2, x2-x-4)
End Function
Function SaveFile(aField, aFilename)
   Dim FSO, TS
   Set FSO = _
      server.CreateObject( _
      "Scripting.FileSystemObject")
   Set TS = FSO.CreateTextFile(aFilename, True,
      False)
   TS.Write GetFileData(aField)
   TS.Close
   Set TS = Nothing
   Set FSO = Nothing
End Function
%>
```

—Matt Hart, Tulsa, Okla.

C#
Level: Beginning

Format a Number in a String
If you want to format a number in a string so it's in hexadecimal format rather than decimal, use the Format method for the data type:

```
int I = 123;

Console.WriteLine (
   "decimal i = " + i +
   " hexadecimal i = " +
   int.Format(i, "x8"));
```

The number after the *x* tells Format how many digits to display, adding zeros to pad out the number of digits you specify. Here's the output from this line:

```
decimal i = 123
hexadecimal i = 0000007b
```

—Andy Harding, Kirkland, Wash.

VB3, VB4, VB5, VB6
Level: Beginning

Loop Using GetTickCount or timeGetTime
This TickDiff function returns the difference of two calls to GetTickCount or timeGetTime, taking into account things such as VB's two's complement as well as wrapping by the operating system:

```
#If Win32 Then
   Private Declare Function GetTickCount _
      Lib "kernel32" () As Long
#Else
   Private Declare Function GetTickCount _
      Lib "User" () As Long
#End If

Public Sub SampleLoop()
   Dim TickStart As Currency
   TickStart = GetTickCount()
   ' Loop for 5 seconds
   Do While TickDiff(TickStart, GetTickCount()) _
      < 5000
      ' loop code here
   Loop
End Sub

Public Function TickDiff( _
   ByVal TickStart As Currency, _
   ByVal TickEnd As Currency) As Long

   ' CCur(2 ^ 32)
   Const TwoToThe32nd As Currency = 4294967296@

   ' Handle two's complement for values larger than
   ' 2147483647&
   If TickStart < 0 Then
      TickStart = TickStart + TwoToThe32nd
   End If
   ' Handle two's complement AND the case where
   ' timeGetTime/GetTickCount wraps at (2 ^ 32)ms,
   ' or ~49.7 days:
   If (TickEnd < 0) Or (TickEnd < TickStart) Then
      TickEnd = TickEnd + TwoToThe32nd
   End If
   ' Return the result
   TickDiff = TickEnd - TickStart
End Function
```

—Monte Hansen, Ripon, Calif.

VS.NET
Level: Beginning

Work With Miscellaneous Files
Visual Studio.NET has a miscellaneous files feature that allows you to create links to other files in Solution Explorer. The files are not added to your project or copied to the project folder. This comes in handy if you want to refer to some reference document while working on a project, such as a readme file, or when you're writing API declarations. You can use the Find in Files features of VS.NET to locate the .h (C/C++ Header file) that has the functions and constants declared. Double-click on the file in the results to add it to your miscellaneous files.

The miscellaneous files will contain links to them from Solution Explorer the next time you open the project. To enable this, you need to turn on the miscellaneous files option and set the limit for how many files should be remembered per solution. To do this, you select Tools | Options | Environment | Documents and check the Show Miscellaneous Files feature. Then type in the number of files to remember (between 0 and 256 per solution).

—Bill McCarthy, Barongarook, Victoria, Australia

VB6, SQL Server 7.0 and up
Level: Advanced

Prevent SQL Server From Returning Record Counts
If you're calling a stored procedure from a Microsoft Transaction Server (MTS) transaction and that stored procedure performs several actions before completing, SQL Server returns a count of affected records for each action. If an error occurs after the first record count is returned, the MTS transaction won't acknowledge the error because it sees the record count returned as a success message. To alleviate this problem, put "SET NOCOUNT ON" at the top of the stored procedure. Setting this option prevents SQL Server from returning record counts.

This example will *not* return an error to an MTS transaction:

```
USE Northwind
GO
CREATE PROCEDURE sp_TestMTSNoError
AS
/*
** This query returns a count of affected record equal to 1.
*/
UPDATE Orders
SET EmployeeID = 5
WHERE OrderID = 10248
/*
** This query generates a 'Divide by Zero' error.
*/
SELECT 1/0
GO
/*
** Running this in Query Analyzer will show an error.
*/
EXECUTE sp_TestMTSNoError
Returns:
(1 row(s) affected)
Server: Msg 8134, Level 16, State 1, Procedure
sp_TestMTSNoError, Line 16
Divide by zero error encountered.
```

An MTS transaction won't see the error message returned by this query unless you apply the "SET NOCOUNT ON" option.

—Jason Rein, Thompson's Station, Tenn.

VB4, VB5, VB6
Level: Beginning

Tiling Made Easy
Use this method whenever you need a tiled background on your forms. All you need is an Image control with its Visible property set to False, and a graphic in its Picture property. Set the form's AutoRedraw property to False and place this code inside the Form_Paint event:

```
Private Sub Form_Paint()
   Dim X As Single
   Dim Y As Single

   For Y = 0 To Me.ScaleHeight Step Image1.Height
      For X = 0 To Me.ScaleWidth Step Image1.Width
         Me.PaintPicture Image1.Picture, X, Y
      Next X
   Next Y
End Sub
```

This code tiles within a PictureBox control as well. Simply replace Me with the name of the PictureBox.

—Brian McDonald, Covington, Ky.

XML, .NET beta 1 and up
Level: Beginning

☆☆☆☆☆ Five Star Tip

Read XML Documents Efficiently

In applications where you need to read XML documents in the most efficient manner possible, consider using the XmlTextReader class rather than the XmlDocument class. The XmlTextReader is a stream that allows tokens found in the source XML document to be read in a forward-only manner without loading the entire XML document into memory as with the DOM. Although at first glance this might seem similar to Simple API for XML (SAX), the XmlTextReader exposes a pull model rather than the push model found in SAX. The pull model offers many performance benefits. Using the XmlTextReader in an ASPX page is as simple as calling the read() method (you must reference the System.Xml namespace):

```
XmlTextReader reader = new
      XmlTextReader(Server.MapPath("myfile.xml"));
while (reader.Read()) {
   if (reader.NodeType == XmlNodeType.Element) {
      Response.Write("Found an Element!<br />");
      if (reader.HasAttributes()) {
         while (reader.MoveToNextAttribute()) {
            Response.Write(
                  "&nbsp;&nbsp;Found an Attribute!<br />");
         }
      }
   }
}
// Make sure you close the stream to prevent file locking
reader.Close();
```

—Dan Wahlin, Chandler, Ariz.

VB5, VB6, SQL Server 7.0 and up
Level: Intermediate

Execute a Temporary SQL Stored Procedure

If a user doesn't have permission to create a stored procedure in SQL Server (version 7.0 or later), he or she can still use ADO to create a temporary stored procedure and execute it from VB:

```
Dim cmd As ADODB.Command
Dim conn As ADODB.Connection
Set conn = New ADODB.Connection
Set cmd = New ADODB.Command
' Replace the connection string values below as
' required.
conn.Open "Provider=SQLOLEDB;Data" & _
   " Source=MyDataSource;Initial" & _
   " Catalog=InitialCatalog; User" & _
   " ID=userID;Password=password"
strCmd = "SELECT Customers.CustomerID, OrderID," & _
   " ContactName INTO #tempTable FROM Customers" & _
   " INNER Join Orders ON Orders.CustomerID =" & _
   " Customers.CustomerID"
cmd.ActiveConnection = conn
cmd.CommandText = strCmd
cmd.CommandType = adCmdText
cmd.Prepared = True
cmd.Execute
```

Using the Prepared property causes SQL Server to create and store a temporary stored procedure in the tempdb database, in a technique known as Prepared/Execute. The command text can contain most things you could put in a SQL stored procedure.

—Parthasarathy Mandayam, Bellevue, Wash.

VS.NET
Level: Beginning

Use #region and #endregion to Organize Code

The #region and #endregion preprocessor directives define blocks of code you can expand or collapse in the Visual Studio editor:

```
#region MenuHandlers
   private void OnNew (object sender, EventArgs e)
   {
      Invalidate ();
   }

   private void OnExit (object sender, EventArgs e)
   {
      Close ();
   }

   private void OnOpen (object sender, EventArgs e)
   {
      Close ();
   }

   private void OnSave (object sender, EventArgs e)
   {
      Close ();
   }
#endregion
```

This code defines a region called MenuHandlers. You can expand or collapse this node in Visual Studio using the Visual Studio Editor Outline feature. This feature lets you show only the code you're working with; it hides the rest in a class.

—Bill Wagner, Manchester, Mich.

VB4, VB5, VB6
Level: Intermediate

Déjà Queue

I especially enjoyed the "Quick and Easy Queue" tip in the 11th edition of "101 Tech Tips for VB Developers" [*Visual Basic Programmer's Journal* March 2001]. You can use the technique for an "undo" menu/toolbar option. Use a collection instead of a listbox to reduce overhead. Support for both LIFO and FIFO is also included:

```
Public Queue As New Collection
Public Const Q_LIFO = 0
Public Const Q_FIFO = 1

Public Sub Enqueue(QueueItem As Variant)
   Queue.Add QueueItem
End Sub

Public Function Dequeue(Optional Mode As Long) As Variant
   Dim Position as Long

   If Queue.Count > 0 Then
      If Mode = Q_LIFO Then
         Position = Queue.Count
      Else
         Position = 1
      End If
      Dequeue = Queue(Position)
      Queue.Remove Position
   Else
      Dequeue = Null
   End If
End Function
```

—Brian Ray, Rockford, Ill.

VB6
Level: Advanced

Open a ToolBar Dropdown Menu

VB6 introduced a new Style=5-tbrDropDown of the Button object for the ToolBar control. In this case, you can add one or more ButtonMenu objects to the current Button. Unfortunately, you can only open the dropdown menu by clicking on the drop-down arrow; in other words, it has no built-in functionality for opening a dropdown menu from code. Here's one function, ShowPopUpMenu, that lets you open a dropdown menu from any place in your source code:

```
Private Type POINTAPI
  x As Long
  y As Long
End Type

Private Declare Function GetCursorPos Lib "user32" _
    (lpPoint As POINTAPI) As Long
Private Declare Function ClientToScreen Lib _
    "user32" (ByVal hWnd As Long, _
    lpPoint As POINTAPI) As Long
Private Declare Function SetCursorPos Lib "user32"_
    (ByVal x As Long, ByVal y As Long) As Long
Private Declare Function ShowCursor Lib "user32" _
    (ByVal bShow As Long) As Long
Private Declare Sub mouse_event Lib "user32" ( _
    ByVal dwFlags As Long, ByVal dx As Long, _
    ByVal dy As Long, ByVal cButtons As Long, _
    ByVal dwExtraInfo As Long)

Private Sub ShowPopUpMenu(TB As Toolbar, _
   IndexOfButton%)
   Const MOUSEEVENTF_LEFTDOWN = &H2
   Const MOUSEEVENTF_LEFTUP = &H4
   Dim Pt As POINTAPI, oldPt As POINTAPI
   With TB.Buttons(IndexOfButton)
      If Not (.Style = tbrDropdown) Then Exit Sub
      Call GetCursorPos(oldPt)
      Call ClientToScreen(TB.hWnd, Pt)
      Call ShowCursor(False)
      Call SetCursorPos(Pt.x + ((.Left + .Width) / _
         Screen.TwipsPerPixelX) - 1, _
         Pt.y + ((.Top + .Height ) 2) / _
         Screen.TwipsPerPixelY))
   End With
   Call mouse_event(MOUSEEVENTF_LEFTDOWN, _
      0, 0, 0, 0)
   Call mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0)
   Call SetCursorPos(oldPt.x, oldPt.y)
   Call ShowCursor(True)
End Sub
```

For example, if you need to open a dropdown menu when a user clicks on a main part of any button, use this source code:

```
Sub ToolBar1_ButtonClick(ByVal Button As _
   MSComctlLib.Button)
   Call ShowPopUpMenu(ToolBar1, Button.Index)
End Sub
```

—Vladimir Olifer, Staten Island, N.Y.

VB3, VB4, VB5, VB6
Level: Intermediate

Replicate Character Patterns

VB's String function is useful to fill a large string with a specific character. Occasionally, you might need to fill a string with a repeating set of characters. If you're using a small database held inside a string, for example, you might want to set default values for some of the fields.

When the need does arise, you can use VB's Mid statement to handle the task:

```
Dim Data As String
Const Rep = "ABCD"

Data = Rep & Space$(1000)
Mid$(Data, Len(Rep) + 1) = Data
```

These last two statements do a significant amount of work. The first line allocates the required memory, and the second fills that memory with character data—that's pretty good for only two lines of code. Not surprisingly, this step is quick, even for large strings, and it provides better functionality than VB's regular String function:

```
Public Function Replicate (ByVal Number As Long, _
   ByVal Pattern As String) As String
   ' Returns PATTERN replicated in a string NUMBER times.
   ' Number  = Number of replications desired
   ' Pattern = Character pattern to replicate
   Dim LP As Long
   Dim sRet As String
   If Number > 0 Then
      LP = Len(Pattern)
      If LP > 1 Then
         sRet = Pattern & Space$((Number - 1) * LP)
         If Number > 1 Then
            Mid$(sRet, LP + 1) = sRet
         End If
      Else
         sRet = String$(Number, Pattern)
      End If
   End If
   Replicate = sRet
End Function
```

—Larry Serflaten, Monticello, Minn.

VS.NET
Level: Intermediate

Examine IL

When you compile a VB.NET or C# project, the code is compiled to Microsoft Intermediate Language (MSIL). You can view the IL code in ILDasm.exe. Launch ILDASM from the VS.NET IDE to view the compiled IL for your code quickly. First, ensure that ILDASM is installed—it should be located in the .NET Framework tools directory. If it is not in there, run Setup again and make sure you install the .NET SDK components. Select External Tools from the Tools menu. Add an entry for ILDASM and specify these parameters:

```
Command: the full path to ildasm.exe
Arguments: $(TargetPath)
```

When you want to look at the IL for your program, simply build it and click on your ILDASM entry in the Tools menu. To dump the IL to a text file, specify this parameter:

```
Arguments:$(TargetPath) /out=$(TargetDir)$(TargetName).il
```

—Bill McCarthy, Barongarook, Victoria, Australia

VB3, VB4, VB5, VB6
Level: Advanced

Boundless Array Indexing

Use this function for getting the 10th next element from an array with "circular" contents such as "Monday," "Tuesday," and so on:

```
intCurrent = WrapIndex(Index:=intCurrent,
   Move:=+10, UpperBound:=6)
```

You don't have to think about passing the end of the array as long as that's what you want to do:

```
Public Function WrapIndex(ByVal Index As Long, _
   ByVal Move As Long, _
   ByVal UpperBound As Long, _
   Optional ByVal LowerBound As Long) As Long
   ' Function for incrementing an index past UB
   ' and starting over from LB, or the other way
   ' around.
   ' If LowerBound is omitted, 0-base is assumed.
   Dim lngCount As Long

   If UpperBound = LowerBound Then
      WrapIndex = LowerBound
   Else
      ' Swap UpperBound and LowerBound if needed
      If LowerBound > UpperBound Then
         LowerBound = LowerBound Xor UpperBound
         UpperBound = LowerBound Xor UpperBound
         LowerBound = LowerBound Xor UpperBound
      End If

      ' number of elements in range
      lngCount = UpperBound - LowerBound + 1

      ' Move Index inside of range
      ' LowerBound...UpperBound if needed
      If Index < LowerBound Then
         Index = UpperBound - ((LowerBound -_
            Index) Mod lngCount) + 1
      ElseIf Index > UpperBound Then
         Index = LowerBound + ((Index -_
            UpperBound) Mod lngCount) - 1
      End If

      ' Move to the new index
      Select Case Move
         Case Is > 0
            WrapIndex = (Index - LowerBound + _
               Move) Mod lngCount + LowerBound
         Case Is < 0
            WrapIndex = (Index - LowerBound + _
               lngCount - Abs(Move Mod lngCount)) _
               Mod lngCount + LowerBound
         Case 0
            WrapIndex = Index
      End Select
   End If
End Function
```

—André Lomøy, Oslo, Norway

VB4/32, VB5, VB6, VBS, SQL 7.0 and up
Level: Intermediate

Avoid Zero-Length String Parameter Failures

Have you ever had a zero-length string parameter fail when attempting to execute a stored procedure from ADO? You'll find this warning buried in the ADO documentation under the Append method for the Parameters collection: "If you select a variable-length data type, you must also set the Size property to a value greater than zero." This refers to any parameter passed as type adLongVarChar, adLongVarWChar, adVarChar, or adVarWChar. I use adVarWChar to send a parameter to a SQL Server stored procedure expecting a varchar(N), so I've had problems with zero-length strings. If you pass a zero-length string and use VB's Len function to retrieve its length, an error results when the length is passed as 0.

I wrote a simple function, LenStringParameter, to return a length of 1 instead. Place the function in a module to make it available from anywhere in your app:

```
Function LenStringParameter(strParam As String) As Long
' From the ADO BOL: "If you select a variable-length
' data type, you must also set the Size property to a
' value greater than zero."
' The length must be passed as 1 even if the string
' is empty or Null.
LenStringParameter = IIf(Len(strParam) = 0, 1, _
   Len(strParam))
End Function
```

Test this function by substituting LenStringParameter for Len wherever needed:

```
Public Function TestStringParameter() as String
Dim strMyTestValue As String
Dim intOtherValue As Integer
strMyTestValue = TextMyKeyValue.Text
' Read from a text field, or assign directly.
intOtherValue = 1
Dim cmdADO As ADODB.Command: Set cmdADO = _
   New ADODB.Command
With cmdADO
.ActiveConnection = strConnection
' Your connection string or connection here.
.CommandType = adCmdStoredProc
.CommandText = "spReturnMyAnswer"
.Parameters.Append .CreateParameter("MyTestValue", _
   adVarWChar, adParamInputOutput, _
   LenStringParameter(strMyTestValue), _
   strMyTestValue)
.Parameters.Append .CreateParameter("OtherValue", _
   adInteger, adParamInputOutput, _
   Len(intOtherValue), intOtherValue)
.Execute
' Pick up the return values.
strMyTestValue = .Parameters("MyTestValue").Value
intOtherValue = .Parameters("OtherValue").Value
End With
Set cmdADO = Nothing
TestStringParameter = strMyTestValue
End Function
```

A word of caution: If you expect an adParamInputOutput type parameter's return value to be larger than the size going in, don't use LenStringParameter. You'll receive truncated data. In other words, if LenStringParameter returns 20, and 20 is sent as the size of an adParamInputOutput type parameter, a maximum of 20 characters will be returned, even if the stored procedure sets the value of the OUTPUT parameter to a value longer than 20 characters. Instead, set the size to the maximum allowed by the stored procedure. If the stored procedure expects a varchar(25) OUTPUT, send the length as 25. If it returns less than 25, the extra space will be discarded.

—Jake Mireles, Houston

VB4/32, VB5, VB6
Level: Intermediate

Convert a VB CommandButton Into a Picture Button at Run Time

I first used this function in the VB4 days when CommandButtons didn't have a graphical Style property. It's still useful because you can't set the Style property at run time. You can use this technique to produce many different styles of intrinsic controls:

```
Private Declare Function GetWindowLong Lib "user32" _
   Alias "GetWindowLongA" (ByVal hWnd As Long, _
   ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib "user32" _
   Alias "SetWindowLongA" (ByVal hWnd As Long, _
   ByVal nIndex As Long, ByVal dwNewLong As Long) _
   As Long
Private Declare Function SendMessage Lib "user32" _
   Alias "SendMessageA" (ByVal hWnd As Long, _
   ByVal wMsg As Long, ByVal wParam As Long, _
   ByVal lParam As Long) As Long

Private Const BS_BITMAP As Long = &H80&
Private Const BS_ICON As Long = &H40&
Private Const BS_TEXT As Long = 0&
Private Const BM_GETIMAGE As Long = &HF6
Private Const BM_SETIMAGE As Long = &HF7
Private Const IMAGE_BITMAP As Long = 0&
Private Const IMAGE_ICON As Long = 1&
Private Const GWL_STYLE As Long = (-16&)
Private Const GWL_EXSTYLE As Long = (-20&)

Public Sub SetButtonGlyph(Button As CommandButton, _
   Picture As StdPicture)
   Dim dwStyle As Long
   Dim hImage As Long
   Dim ImageType As Long

   ' Get the button style
   dwStyle = GetWindowLong(Button.hWnd, GWL_STYLE)

   If Picture Is Nothing Then
      ' Clear the graphic style, add the text style
      dwStyle = (dwStyle Or BS_TEXT) And _
         Not (BS_BITMAP Or BS_ICON)
      If (dwStyle And BS_BITMAP) <> 0 Then
         Call SendMessage(Button.hWnd, _
            BM_SETIMAGE, IMAGE_BITMAP, 0&)
      ElseIf (dwStyle And BS_ICON) <> 0 Then
         Call SendMessage(Button.hWnd, _
            BM_SETIMAGE, IMAGE_ICON, 0&)
      End If

      ' Update style bits & redraw
      SetWindowLong Button.hWnd, GWL_STYLE, dwStyle
      Button.Refresh

   Else
      ' Remove mutually exclusive bits
      dwStyle = dwStyle And _
         Not (BS_BITMAP Or BS_ICON Or BS_TEXT)
      Select Case Picture.Type
      Case vbPicTypeIcon
         dwStyle = dwStyle Or BS_ICON
         ImageType = IMAGE_ICON
      Case vbPicTypeBitmap
         dwStyle = dwStyle Or BS_BITMAP
         ImageType = IMAGE_BITMAP
      End Select

      ' Handle of image to attach to button.
      hImage = Picture.Handle

      ' Change the style of the button
      Call SetWindowLong(Button.hWnd, GWL_STYLE, _
         dwStyle)
      ' Add or remove the glyph
      Call SendMessage(Button.hWnd, BM_SETIMAGE, _
         ImageType, hImage)
   End If
End Sub
```

—Monte Hansen, Ripon, Calif.

VB3, VB4, VB5, VB6
Level: Beginning

Object Properties as Parameters are ByVal Only

In VB, we can use a function/sub call to return results by passing parameters by reference (although it's generally a bad idea). Be aware that if you use an object's properties as parameters, the result might not be what you expected. For example, create a simple form application with a textbox and command button, then run this:

```
Private Sub Command1_Click()
   Dim szMsg As String
   szMsg = "Before: """ & Text1.Text & """" & vbCrLf
   Call testStr(Text1.Text)
   szMsg = szMsg & "After: """ & Text1.Text & """"
   MsgBox szMsg
End Sub

Private Sub testStr(aszText As String)
   aszText = "Text string changed"
End Sub
```

If you use an object's property directly as a parameter, that property won't be updated. The reason is simple: VB copies the property into a temporary memory location and passes that memory as the parameter into the function/sub. VB doesn't copy the memory back to the object's property after the call, so any changes you make are lost.

—David Chu, Calgary, Alberta

VB3, VB4, VB5, VB6
Level: Beginning

App.Path is Inconsistent

The path returned by App.Path is inconsistent. If the program is running in a root directory, the path will have a backslash on the end. Otherwise, it won't. The solution is to write one or two wrapper functions to ensure a path has a backslash on the end:

```
Public Function NormalizePath( _
   ByVal strPath As String) As String
   ' If the path doesn't have a slash at the end,
   ' add one.
   If Right$(strPath, 1) = "\" Then
      NormalizePath = strPath & "\"
   Else
      NormalizePath = strPath
   End If
End Function

Public Function AppPath() As String
   ' Return the normalized App.Path ...
   AppPath = NormalizePath(App.Path)
End Function
```

Now you get a consistent App.Path easily by calling the AppPath function.

—Chris Hynes, Fort Washington, Md.

VB.NET
Level: Intermediate

Use Int16, Int32, and Int64 for API Declarations
When declaring API functions in VB.NET, use Int16, Int32, and Int64 rather than Short, Integer, and Long, respectively, to avoid possible confusion with VB6 declarations. Note: In VB6, Long is 32-bit and Integer is 16-bit, whereas in VB.NET, Long is 64-bit and Integer is 32-bit.

—Bill McCarthy, Barongarook, Victoria, Australia

VB5, VB6
Level: Intermediate

Make the Background of Your RichTextBox Controls Transparent
If you plan to *require Windows 2000* for your application, you can make your standard VB RichTextBox control 100-percent transparent with a few simple API calls. To try this tip, create a new project (or use an existing one), add a RichTextBox control, and add this code and these declarations in a standard module:

```
Option Explicit

' Win32 APIs.
Private Declare Function GetWindowLong _
   Lib "user32" Alias "GetWindowLongA" _
   (ByVal hWnd As Long, _
   ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong _
   Lib "user32" Alias "SetWindowLongA" _
   (ByVal hWnd As Long, ByVal nIndex As Long, _
   ByVal dwNewLong As Long) As Long
Private Declare Function SetWindowPos Lib "user32" _
   (ByVal hWnd As Long, ByVal hWndInsertAfter As _
   Long, ByVal X As Long, ByVal Y As Long, _
   ByVal cx As Long, ByVal cy As Long, _
   ByVal wFlags As Long) As Long

' Style bits.
Private Const GWL_EXSTYLE As Long = (-20)
Private Const WS_EX_TRANSPARENT As Long = &H20

' Force total redraw that shows new styles.
Private Const SWP_FRAMECHANGED = &H20
Private Const SWP_NOMOVE = &H2
Private Const SWP_NOZORDER = &H4
Private Const SWP_NOSIZE = &H1

Public Function Transparent(ByVal hWnd As Long, _
   Optional ByVal Value As Boolean = True) As _
   Boolean

   Dim nStyle As Long
   Const swpFlags As Long = _
      SWP_FRAMECHANGED Or SWP_NOMOVE Or _
      SWP_NOZORDER Or SWP_NOSIZE

   ' Get current style bits.
   nStyle = GetWindowLong(hWnd, GWL_EXSTYLE)
   ' Set new bits as desired.
   If Value Then
      nStyle = nStyle Or WS_EX_TRANSPARENT
   Else
      nStyle = nStyle And Not WS_EX_TRANSPARENT
   End If
   Call SetWindowLong(hWnd, GWL_EXSTYLE, nStyle)
   ' Force redraw using new bits.
   SetWindowPos hWnd, 0, 0, 0, 0, 0, swpFlags
   ' Make sure new style took.
   Transparent = _
      (GetWindowLong(hWnd, GWL_EXSTYLE) = nStyle)
End Function
```

You can use this function to toggle the transparency of your RichTextBox controls at will:

```
Private Sub Check1_Click()
   Call Transparent(RichTextBox1.hWnd, _
      (Check1.Value = vbChecked))
End Sub
```

To be on the safe side, check the OS version before making these calls, as the effects can be rather unpleasant in the wrong environment.

That's it! A simple call to Get/SetWindowLong retrieves the current extended style bits and adds the standard TRANSPARENT style so the window becomes transparent. Note, if you change the style *after* the control is visible, you need to force the screen to repaint to see the effect.

—John Cullen, Pedroucos, Portugal

VB4, VB5, VB6
Level: Beginning

Duck the Modal Form PopupMenu Bug
Microsoft confirms this bug: If an application contains at least two forms, and one of those forms is displayed modally using a PopupMenu on another form, a PopupMenu on the modal form won't be displayed. Knowledge Base article *Q167839 - BUG: PopupMenu on Modal Form Not Displayed* suggests using a Timer control as a workaround for this problem. My solution sets a flag variable in the first form's menu procedure, which is then acted upon after the popup is dismissed.

Start a new Standard EXE project. Form1 is added by default. Add another form (Form2) to the project. On Form1, create an invisible menu (mnuFile) with the caption "File" that has a submenu (mnuOpen) with the caption "Open". On Form2, create an invisible menu (mnuEdit) with the caption "Edit" that has a submenu (mnuFind) with the caption "Find". Then add this code to Form1:

```
Private bShowForm2 as Boolean

Private Sub Form_Click()
   PopupMenu mnuFile
   If bShowForm2 Then
      bShowForm2 = False
      Form2.Show vbModal
   End If
End Sub

Private Sub mnuOpen_Click()
   bShowForm2 = True
End Sub
```

Add this code to Form2:

```
Private Sub Form_Click()
   PopupMenu mnuEdit
End Sub
```

Press F5 to run the program. Click on Form1 to display the File PopupMenu. Select Open to show Form2 modally. Click on Form2 to display the Edit PopupMenu.

—Peter Gabris, Marietta, Ga.

For even more tricks and tips go to
www.vbpj.com or www.vcdj.com

VB3, VB4, VB5, VB6
Level: Beginning

Open Namespace Objects From VB

As you probably know, you can open an Explorer window on a directory from VB by using the intrinsic Shell function:

```
Dim TaskId As Long
TaskId = Shell("Explorer c:\", vbNormalFocus)
```

But what about items in the namespace that don't correspond to physical file system objects, such as the Printers and Task Scheduler folders—or My Computer itself, for that matter?

Here's a little-known trick: You can pass an argument to Explorer indicating the namespace object by its GUID, and Explorer dutifully opens it for you. For example, to open Scheduled Tasks, you could use this:

```
Dim TaskId As Long
Dim ShellCmd As String
ShellCmd = "Explorer ::{20D04FE0-" & _
   "3AEA-1069-A2D8-08002B30309D}" & _
   "\::{D6277990-4C6A-11CF-8D87-" & _
   "00AA0060F5BF}"
TaskId = Shell(ShellCmd, vbNormalFocus)
```

Here's a list of the most common namespace objects and the equivalent "paths" to pass to Explorer in the Shell command; this information comes straight from the Windows Registry, which you can also search for other namespace objects. Now you can open any of these objects right from within VB:

```
My Computer
::{20D04FE0-3AEA-1069-A2D8-08002B30309D}

Network Neighborhood
::{208D2C60-3AEA-1069-A2D7-08002B30309D}

Recycle Bin
::{645FF040-5081-101B-9F08-00AA002F954E}

Task Scheduler
::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\::{D6277990-4C6A-11CF-8D87-00AA0060F5BF}

Printers
::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\::{2227A280-3AEA-1069-A2DE-08002B30309D}

Control Panel
::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\::{21EC2020-3AEA-1069-A2DD-08002B30309D}

Dial-up Networking
::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\::{a4d92740-67cd-11cf-96f2-00aa00a11dd9}

Web Folders
::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\::{BDEADF00-C265-11D0-BCED-00A0C90AB50F}
```

—Jason Fisher, Dallas

VS.NET
Level: Beginning

Customize Toolbox Icons

You can create your own custom bitmap to give your UserControl or component its own unique toolbox icon. Simply add a 16-by-16 bitmap to your project and give it the same name as the component; for example, MyComponent.bmp. Set its Build action to Embedded Resource.

—Bill McCarthy, Barongarook, Victoria, Australia

VB3, VB4, VB5, VB6
Level: Beginning

Keep Selected Areas in Grid Read-Only

I needed to lock a row on a grid to show totals as well as a percentage row that needed to remain read-only to the user. I didn't want to add another grid with a single row for totals as it didn't have the flexibility I needed. After some Web searching, I found this to be a common issue. After much hair-pulling and many caffeinated beverages, I discovered this simple and basic answer in one line of code. Enjoy:

```
Private Sub MyDBGrid_KeyPress(KeyAscii As Integer)
   ' Whatever row you want to be "locked"
   If MyDBGrid.Row = MyTotalsRow Then KeyAscii = 0
End Sub
```

—Joe Johnston, Chesapeake, Va.

VB6, VBS
Level: Beginning

Quick Split

When you use the Split function from VB6 or VBScript, sometimes you need only a single value and not the whole array. To do this, you can reference the element you need right after the Split statement like this:

```
Split(myVar, myDelim)(1)
```

This statement retrieves the second element of the array, element 1.

—Judah Reeves, San Diego

VB4, VB5, VB6
Level: Beginning

Zoom Continuously in Your Image-Processing Apps

This code demonstrates how fast you can zoom into images using VB's form PaintPicture method. Start a new project containing two forms named frmClip and frmPicture. frmPicture contains a Shape control named shpRectangle, an Image control named pic, and all the code. frmClip contains no code, but it's the target of the clipped image within shpRectangle as it is dragged over pic:

```
' frmPicture
Option Explicit
Private mlTop As Long
Private mlLeft As Long
Private mlRight As Long
Private mlBottom As Long

Private Sub Form_Load()
   Me.ScaleMode = vbTwips
   With pic
      .BorderStyle = 0  none
      .Move 0, 0
      .Picture = LoadPicture( _
         "C:\Anderson\Imaging\Images\address.bmp")
      .ZOrder vbSendToBack
   End With
   With shpRectangle
      .Shape = 0 ' Rectangle
      .BorderStyle = 1 ' solid
      .BorderWidth = 2
      .DrawMode = 13 'Copy pen
      .Visible = False
   End With
   frmClip.ScaleMode = vbTwips
   frmClip.Show
End Sub

Private Sub pic_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    If Button = vbLeftButton Then
       With shpRectangle
          .Visible = False 'reduces flickering
          mlBottom = Y
          mlRight = X

          ' Don't allow clip to include non-picture
          ' region
          If mlBottom > pic.Height Then _
             mlBottom = pic.Height
          If mlRight > pic.Width Then _
             mlRight = pic.Width
          If mlBottom < pic.Top Then _
             mlBottom = pic.Top
          If mlRight < pic.Left Then _
             mlRight = pic.Left

          ' Swap top/bottom as necessary
          If mlBottom < mlTop Then
             .Top = mlBottom
             .Height = mlTop - mlBottom
          Else
             .Top = mlTop
             .Height = mlBottom - mlTop
          End If

          ' Swap left/right as necessary
          If mlRight < mlLeft Then
             .Left = mlRight
             .Width = mlLeft - mlRight
          Else
             .Left = mlLeft
             .Width = mlRight - mlLeft
          End If

          .Visible = True

          DoEvents ' Allow rectangle to draw
          frmClip.PaintPicture pic.Picture, 0, 0, _
             frmClip.ScaleWidth, _
             frmClip.ScaleHeight, _
             .Left - pic.Left, .Top - pic.Top, _
             .Width, .Height
       End With
    End If
End Sub

Private Sub pic_MouseDown(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    If Button = vbLeftButton Then
       mlTop = Y
       mlLeft = X
    End If
End Sub

Private Sub pic_MouseUp(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    shpRectangle.Visible = False
End Sub
```

—Graeme Anderson, Blackburn, Australia

VB5, VB6
Level: Intermediate

Merge VBL Files Into Your Registry

Use this tip when developing ActiveX components and testing ActiveX OCXs that require license files. These Registry entries allow you to merge VBL file contents into the Registry. You add three context menu options for VBL files: Merge (into the Registry), Edit, and Print:

```
REGEDIT4

[HKEY_CLASSES_ROOT\.vbl]
@="VisualBasic.VBLFile"

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile]
@="Visual Basic Control License File"

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\DefaultIcon]
@="NOTEPAD.EXE,1"

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell]

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell\edit]
@="&Edit"

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell\edit\command]
@="NOTEPAD.EXE \"%1\""

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell\open]
@="Mer&ge"

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell\open\command]
@="regedit.exe \"%1\""

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell\print]

[HKEY_CLASSES_ROOT\VisualBasic.VBLFile\shell\print\command]
@="NOTEPAD.EXE /P \"%1\""
```

Editor's Note: All the usual warnings apply, of course, when running scripts against your Registry.

—Tom Sweet, Marietta, Ga.

VB4, VB5, VB6
Level: Beginning

Use the Keyboard for Extra-Fine Sizing and Positioning

If you have a sensitive mouse and/or many objects you need to position carefully on a form, getting everything right can be a real pain—especially when you click on an object simply to alter some of its properties and move it accidentally. Here's the answer: Use the "Lock Controls" option under the Format menu to lock the position of a form's controls so you can't change control positions accidentally when clicking on the controls. But now you can't use the mouse to reposition a control without unlocking everything. However, you *can* use the keyboard. Simply select the control whose position you want to change and combine the Ctrl key and arrow keys to move the control, or the Shift key and arrow keys to resize it. The grid spacing you've configured (Tools | Options | General) controls the move/size extent per arrow press.

—John Cullen, Pedroucos, Portugal

VB4, VB5, VB6
Level: Intermediate

Override the Err Object

Believe it or not, you can override VB's built-in Err object to add functionality the Err object doesn't offer. Create a class called CError, then add this property procedure:

```
Public Property Get Number() As Long
    Number = VBA.Err.Number
End Property
```

Now create a BAS module and add this code:

```
Public Function Err() As CError
    Static oErr As CError
    If oErr Is Nothing Then
        Set oErr = New CError
    End If
    Set Err = oErr
End Function
```

Now you can retrieve the custom CError object anywhere in your project that you reference the Err object. Of course, you want to add support for standard properties and methods such as Description, Source, Raise, and so on. You can also add support for other things:

- Err.Line (could return the Erl)
- Err.Log (could log the error to a text file or the event log)
- Err.FullDescription (could wrap the number, source, and description into one nicely formatted string)
- Err.Stack (could return stack trace information)
- Err.Show (could display a nicely formatted message box describing the error)

The nice thing about this approach is that it consolidates all your error-handling code neatly into what appears to be the Err object itself.

—Darin Higgins, Fort Worth, Texas

VB5, VB6
Level: Beginning

Copy an Array Faster, Redux

Simple is usually best. The "Copy an Array Faster" tip in the 11th Edition of "101 Tech Tips for VB Developers" [*Visual Basic Programmer's Journal* March 2001] describes a method that uses a low-level approach to accelerate array copying. VB4 introduced direct assignment to Byte arrays, and VB5 later expanded that capability to include other array types as well.

This code does the same thing as the previous tip, with the same performance improvement:

```
Dim IntArray1() As Integer
Dim IntArray2() As Integer
ReDim IntArray1(1 To 6000000)
ReDim IntArray2(1 To 6000000)
IntArray2 = IntArray1
```

As the old saying goes: "Keep it simple, stupid." The previous tip is still useful for copying *portions* of arrays, but use direct assignment if you need to copy the entire array.

—Dave Doknjas, Surrey, British Columbia

VB5, VB6
Level: Intermediate

Sort Arrays Faster

When an array is declared as a type where each element occupies a lot of memory (such as a complex user-defined type), sorting the array can become unacceptably slow. To speed things up, the data contained within the array shouldn't be moved in memory any more than necessary.

To achieve this, you can set up a second array of integers that contains the indexes of the main array. Your sorting algorithm changes the order of the index array based on comparisons within the main array. When the process is complete, the main array has not been changed but the index array now contains the indexes of the main array in the sorted order. From this point, you can reorder the main array using the index array, which you can then discard:

```
Dim Idx() As Long             ' index array
Dim Data() As EmployeeData ' data array of UDT
```

Compare elements of the main array during sorting:

```
Data(Idx(i)).LastName < Data(Idx(a)).LastName
```

Reorder elements of the index array during sorting:

```
tmp = Idx(a)
Idx(a) = Idx(i)
Idx(i) = tmp
```

Copy the data array to a reference array:

```
Dim Ref() As EmployeeData ' reference array
Ref = Data
```

Populate the data array in the correct order using the reference array and the index array:

```
For i = LBound(Data) To UBound(Data)
   Data(i) = Ref(Idx(i))
Next
```

Although this process can be many times faster than reordering the main array using the sorting algorithm, it can still take a long time. To save more time, you don't need to reorder the main array at all. Instead, whenever you need to access the data in the main array in order, simply refer to its elements using the index array. For example:

```
Text1.Text = Data(Idx(i)).LastName
```

Unfortunately, using the main array with the index array can cause complications. Sometimes you need to reflect changes made to the main array in the index array. This can be difficult because the main array does not contain information about the index array. Therefore, changing key data in the main array requires reindexing to keep things in sync.

—Steele Cheffers, Perth, Western Australia

VB4, VB5, VB6, VBA
Level: Beginning

Use the Format Function for Regional Settings

Although the GetLocaleInfo API can retrieve just about any regional setting you need, VB's own Format function also can be useful for quick-and-dirty answers to some settings. For example, use this code to read the regional setting for the numeric decimal symbol and thousands separators:

```
strDecimal = Format$(0, ".")
strThousands = Mid$(Format$(1000, "0,0"), 2, 1)
```

—John Sevarts, Heerlen, Netherlands

VB4/32, VB5, VB6, VBS
Level: Intermediate

Create a Duplicate Recordset

The Clone method doesn't quite fit the bill when you want to create a duplicate recordset—it gives you two pointers to the same recordset, so any updates or deletes you make in one will be reflected in the other. This isn't always desirable.

You can create a true duplicate recordset easily using ADO 2.5 (or later) while avoiding a second trip to the server. Use the Stream object as an intermediary to hold the necessary XML and pass that XML back into a second Recordset object:

```
Dim rsOne As New ADODB.Recordset
Dim rsTwo As New ADODB.Recordset
Dim oTempStream As New ADODB.Stream
'Assumes rsOne has been populated with data
rsOne.Save oTempStream, adPersistXML
rsTwo.Open oTempStream
```

The catch: The second Recordset will be a client-side cursor. If you want to commit any changes back to your database, you must establish a new connection, set it on the Recordset, and call UpdateBatch.

—Larry Johnson

VB.NET
Level: Beginning

Instantiate an Object Inline

You can instantiate a new instance of an object inline, making your code more compact. This example shows both versions:

```
Imports System

Public Class Author
   Private fName As String
   Private lName As String

   Public Sub New(ByVal fName As String, ByVal lName As _
      String)
      me.fName = fName
      me.lName = lName
   End Sub

   Public ReadOnly Property FullName() As String
      Get
         Return fName & " " & lName
      End Get
   End Property
End Class

Public Class Test
   Public Shared Sub Main()
      'This is the more verbose method
      Dim author As Author = _
         New Author("Jon", "Goodyear")
      Console.WriteLine(author.FullName)

      'This is the less verbose method
      Console.WriteLine( _
         New Author("Jon", "Goodyear").FullName)
   End Sub
End Class
```

—Jonathan Goodyear, Orlando, Fla.

VB3, VB4, VB5, VB6, VBA, VBS
Level: Beginning

Lock Windows 2000 Instantly

Locking an NT workstation has never been easy. Windows 2000 has a new function, LockWorkStation, that can lock the machine instantly with a single API call:

```
Private Declare Function LockWorkStation Lib _
   "user32.dll" () As Long

Call LockWorkStation
```

In fact, because this function requires no parameters, you can reduce the code to a single line, as well as make it callable from 16-bit code, by invoking it through rundll32:

```
Call Shell("rundll32 user32.dll,LockWorkStation", _
   vbNormalFocus)
```

The workstation locks instantly when this line is executed. Here's the equivalent VBS code:

```
Dim WshShell
Set WshShell = CreateObject("WScript.Shell")
WshShell.Run("rundll32 user32.dll,LockWorkStation")
```

—Brian Abernathy, Marietta, Ga.

VB4, VB5, VB6
Level: Beginning

Enable and Disable Frames

I've had problems enabling and disabling frame controls in Visual Basic. If a frame is set to disabled, all the controls within the frame are disabled—but they still *look* enabled. So I created a simple function that loops through all controls, locating those on a specific frame, and enables or disables them as requested:

```
Private Sub EnableFrameControls( _
   fra As Frame, ByVal Enabled As Boolean)

   Dim ctl As Control
   On Error Resume Next
   For Each ctl In Me.Controls
      If ctl.Container Is fra Then
         ctl.Enabled = Enabled
      End If
   Next ctl
   fra.Enabled = Enabled
End Sub
```

—Chris O'Connor, Wantirna South, Victoria, Australia

VB5, VB6
Level: Beginning

Skip Object Declaration Using "With" Keyword

If you want to use an object in the middle of a routine and avoid declaring the object in the routine, simply use this syntax:

```
With New <object classname>
   .<method/property>
End With
```

This is the equivalent of:

```
Dim X as New <object classname>
With X
   .<method/property>
End With
Set X = nothing
```

—Kevin Alons, Salix, Iowa

For even more tricks and tips go to www.vbpj.com or www.vcdj.com

VB5, VB6
Level: Advanced

☆☆☆☆☆ Five Star Tip

Use Argument Arrays With CallByName

VB6 introduced a new built-in function, CallByName(), as a member of VBA.Interaction. It lets you reference an object's method or property by passing its name as an argument. The syntax is:

```
result=CallByName(Object, ProcName, _
   CallType [,ParamArrayArgs])
```

Unfortunately, this function has several restrictions: It's accessible from VB6 only; when an error is raised in an ActiveX procedure called with the CallByName() function from a client, the client always gets "error 440" regardless of the original error number being raised (for details, see Microsoft Knowledge Base article *Q194418 – PRB: CallByName Fails to Return the Correct Error Information*); and the type of the last argument is ParamArray, so you can't create a dynamic list of arguments into one statement. For example, the first and second Call statements of this code don't work:

```
Dim x(1)
x(0) = 1:   x(1) = 2
'--(1) Error (dynamic list):
Call CallByName(Me, "xx", VbMethod, x)
'--(2) Error:
Call CallByName(Me, "xx", VbMethod, Array(1, 2))
'--(3) OK:
Call CallByName(Me, "xx", VbMethod, 1, 2)

Function xx(x1, x2)
'--do something here--
End Function
```

However, you can build your own CallByName function using TypeLib information (TLBINF32.dll) for VB5/6 applications without pointed restrictions:

```
' Required for use in VB5!
Public Enum VbCallType
   VbMethod = 1
   VbGet = 2
   VbLet = 4
   VbSet = 8
End Enum

Public Function CallByNameEx(Obj As Object, _
   ProcName As String, CallType As VbCallType, _
   Optional vArgsArray As Variant)
   Dim oTLI As Object
   Dim ProcID As Long
   Dim numArgs As Long
   Dim i As Long
   Dim v()

   On Error GoTo Handler

   Set oTLI = CreateObject("TLI.TLIApplication")
   ProcID = oTLI.InvokeID(Obj, ProcName)

   If IsMissing(vArgsArray) Then
      CallByNameEx = oTLI.InvokeHook( _
         Obj, ProcID, CallType)
   End If

   If IsArray(vArgsArray) Then
      numArgs = UBound(vArgsArray)
      ReDim v(numArgs)
      For i = 0 To numArgs
         v(i) = vArgsArray(numArgs - i)
      Next i
      CallByNameEx = oTLI.InvokeHookArray( _
         Obj, ProcID, CallType, v)
   End If
Exit Function

Handler:
   Debug.Print Err.Number, Err.Description
End Function
```

You must use this syntax to call the CallByNameEx() function:

```
Call CallByNameEx(Me, "xx", VbMethod, x)
Call CallByNameEx(Me, "xx", VbMethod, Array(1, 2))
Result=CallByNameEx(Me, "xx", VbMethod, x)
```

x is an array containing the same number of elements as the called procedure has parameters. The CallByNameEx() function returns a real error number from a calling procedure. For VB5, you must define the VbCallType Enum used by the third parameter of CallByNameEx(), or use ordinary integers in place of the Enum.

—Vladimir Olifer, Staten Island, N.Y.

VB3, VB4, VB5, VB6
Level: Beginning

Select Case Enhancement

In the January 2000 issue of *Visual Basic Programmer's Journal*, Ron Schwarz wrote a nice article on VB Masonry ("VB Masonry: Applying Mortar to the Bricks"). I have found the need to do multiple tests on dissimilar variables and objects with any failing test causing an action. Multiple embedded If...Then...ElseIf...EndIf statements are awful to look at and troubleshoot. I found that using Select Case does the trick and is easy to read. Consider testing several items before continuing (whether to check during entry or after is another subject). Try this:

```
Private Function okToPost() As Boolean
   ' Assume it's safe to post.
   okToPost = True

   Select Case False
      ' Assume you want your tests to be True
      ' Any tests that evaluate to False will
      ' trigger the case code.
      Case (lvDist.ListItems.Count > 0)
         ' Any items in a listview control?
         MsgBox "No Items Selected", _
            vbInformation, "Post"
         okToPost = False

      Case IsNumeric(fvCheckNumber)
         ' Did the user enter a valid number?
         MsgBox "Invalid Check Number", _
            vbInformation, "Post"
         okToPost = False
         fvCheckNumber.SetFocus

      Case (fvInvoiceAmount = fvCheckAmount)
         ' Does this balance?

      ' More case statements can follow that
      ' evaluate to true or false

   End Select
End Function
```

—Timothy P. Sullivan, Fort Wayne, Ind.

SQL Server 6.5 and up
Level: Intermediate

Use the CASE Statement in a SQL SELECT Clause
SQL Server provides a mechanism for returning different values in a SELECT clause based on Boolean conditions: the CASE statement. This statement resembles Visual Basic's Select Case statement.

The SQL CASE statement has WHEN, THEN, and ELSE clauses along with an END terminator. The syntax is:

```
CASE [expression]
   WHEN [value | Boolean expression] THEN [return value]
   [ELSE [return value]]
END
```

The [expression] is optional and contains a table column or a variable. When you specify [expression] directly after the CASE, you must populate the [value] parameter in the WHEN clause:

```
DECLARE @TestVal int
SET @TestVal = 3

SELECT
   CASE @TestVal
      WHEN 1 THEN 'First'
      WHEN 2 THEN 'Second'
      WHEN 3 THEN 'Third'
      ELSE 'Other'
   END
```

SQL Server compares this value to the expression and when the values match, it returns the THEN clause's [return value]. If none of the WHEN clauses equates to true, SQL Server returns the [return value] in the optional ELSE clause. If the ELSE clause is omitted and no value is matched, NULL is returned.

If you don't specify [expression], you must include the [Boolean expression] in the WHEN clause. This can contain any valid Boolean expression SQL Server allows:

```
DECLARE @TestVal int
SET @TestVal = 5

SELECT
   CASE
      WHEN @TestVal <=3 THEN 'Top 3'
      ELSE 'Other'
   END
```

—Jason Rein, Thompson's Station, Tenn.

VB3, VB4, VB5, VB6
Level: Beginning

Determine the Last Day of the Month
Todd Knudsen submitted a tech tip ["Find the Last Day of a Month," *Visual Basic Programmer's Journal* April 2001] with a function called FindEOM(ADate As Variant) that calculates the end of a month:

```
NextMonth = DateAdd("m", 1, ADate)
FindEOM = NextMonth - DatePart("d", NextMonth)
```

You can accomplish the same thing with one line of code:

```
FindEOM = DateSerial(Year(ADate), Month(ADate) + 1, 0)
```

Setting the day parameter of DateSerial to 0 always returns the day prior to day 1, which is the end of the previous month.

—Barry Garvin, Georgetown, Mass.

VB.NET
Level: Beginning

Initialize Fields in Classes
In VB.NET, you can declare a variable and initialize it on the same line:

```
Dim x as Int32 = 5
```

You can also use this code to initialize fields in a class:

```
Public Class Foo
   Private m_Flag As Int32 = 4

   ...........
End Class
```

Note: In VB.NET, the base class constructor is called before the field is initialized. The compiled code is the equivalent of:

```
Public Class Foo
   Private m_Flag As Int32

   Public Sub New()
      MyBase.New()
      m_Flag = 4
   End Sub

End Class
```

—Bill McCarthy, Barongarook, Victoria, Australia

VB4/32, VB5, VB6
Level: Beginning

Stop the Flickering
This code stops the annoying flicker often seen when you pack an object with data. Test this code with the controls that bother you most:

```
Private Declare Function SendMessage Lib "user32" _
   Alias "SendMessageA" (ByVal hWnd As Long, _
   ByVal wMsg As Long, ByVal wParam As Long, _
   lParam As Any) As Long
Private Const WM_SETREDRAW = &HB

Public Function LockControl(objX As Object, _
   ByVal bLock As Boolean)
   Call SendMessage(objX.hWnd, WM_SETREDRAW, _
      bLock, ByVal 0&)
   If bLock = False Then
      On Error Resume Next
      objX.Refresh
   End If
End Function
```

—Andre Beneke, Reitz, South Africa

VS.NET
Level: Beginning

Increase Your Work Area
VS.NET's auto-hide feature of docked windows enables you to increase your work area. You can select whether a window stays displayed or auto-hides by clicking on the drawing pin icon in the window. One problem you might encounter when using auto-hide windows: When they roll out, they do so on top of the open designer window, thereby hiding part of your form or UserControl. For smaller forms and UserControls, dock the window to the right side of the screen; then when it rolls out, it won't cover the form or UserControl unless either one is extremely large.

—Bill McCarthy, Barongarook, Victoria, Australia

VB4/32, VB5, VB6
Level: Intermediate

Get the Drive Serial Number

You can get the serial number of your hard drive, floppy disk, or CD-ROM easily without any additional ActiveX component. First, start a VB project, add a standard module, and place a Command Button control on the form:

```
'-- Module code
Private Declare Function GetVolumeInformation _
   Lib "kernel32" Alias "GetVolumeInformationA" _
   (ByVal lpRootPathName As String, _
   ByVal pVolumeNameBuffer As String, _
   ByVal nVolumeNameSize As Long, _
   lpVolumeSerialNumber As Long, _
   lpMaximumComponentLength As Long, _
   lpFileSystemFlags As Long, _
   ByVal lpFileSystemNameBuffer As String, _
   ByVal nFileSystemNameSize As Long) As Long

Public Function GetSerialNumber( _
   ByVal sDrive As String) As Long

   If Len(sDrive) Then
      If InStr(sDrive, "\\") = 1 Then
         ' Make sure we end in backslash for UNC
         If Right$(sDrive, 1) <> "\" Then
            sDrive = sDrive & "\"
         End If
      Else
         ' If not UNC, take first letter as drive
         sDrive = Left$(sDrive, 1) & ":\"
      End If
   Else
      ' Else just use current drive
      sDrive = vbNullString
   End If

   ' Grab S/N -- Most params can be NULL
   Call GetVolumeInformation( _
      sDrive, vbNullString, 0, GetSerialNumber, _
      ByVal 0&, ByVal 0&, vbNullString, 0)
End Function

'-- Form code
Private Sub Command1_Click()
   Dim Drive As String
   Drive = InputBox("Enter drive for checking SN")
   MsgBox Hex$(GetSerialNumber(Drive))
End Sub
```

—**Predrag Dervisevic, Krusevac, Yugoslavia**

VB3, VB4, VB5, VB6
Level: Intermediate

Select Areas Within a Graphics Window

Graphics applications sometimes require users to select a rectangular region of a picture or drawing visually. You need to provide a resizing box manipulated by the pointer at run time that only interacts temporarily with the graphics displayed already (download this code).

By assigning vbInvert to the PictureBox DrawMode property before selection dragging, you can restore the background graphics by redrawing the same rectangle. Once the selection dragging completes, mRect contains the selected rectangle coordinates. You can use the same technique to select a circular region or create the "rubber band" effect.

—**James Menesez, Templeton, Calif.**

VB6
Level: Beginning

Read a Complete Text File in One Pass

Typically, you read and process a text file by using a loop and VB's Line Input statement:

```
Do While Not Eof(1)
   Line Input #1, myStringVar$
   ' process the line here
Loop
```

However, you might want to defer processing or keep a copy of all the lines read for repeat processing or selective editing before writing them out again. You can achieve this quite easily by using VB's Get# and Split() statements to read the entire file at once and split it into an array containing all the lines. For example, this function returns the complete contents of a file as a string:

```
Public Function ReadFile(ByVal FileName As String) _
   As String
   Dim hFile As Long
   Dim bBuf() As Byte

   hFile = FreeFile
   Open FileName For Binary Access Read As #hFile
   If LOF(hFile) > 0 Then
      ReDim bBuf(1 To LOF(hFile)) As Byte
      Get #hFile, , bBuf
      Close #hFile
      ReadFile = StrConv(bBuf, vbUnicode)
   End If
End Function
```

This code snippet drops the contents into an array, using the line break (vbCrLf) as a delimiter:

```
   Dim sLines() As String
   Dim sAll As String
   Dim i As Long

   ' Read the contents of some file
   sAll = ReadFile("c:\form1.frm")

   ' Split into individual lines
   sLines = Split(sAll, vbCrLf)
```

You can then process the file as desired; for example, you can search for specific lines:

```
   For i = LBound(sLines) to UBound(sLines)
      If Instr(1, "SomeText", sLines(i), _
         vbTextCompare) Then
         sLines(i) = "SomeOtherText"
      End If
   Next i
```

—**John Cullen, Pedroucos, Portugal**

VB4, VB5, VB6
Level: Beginning

Add Controls to a Project Quickly

VB's Add File dialog supports only a single selection of code modules or OCXs, so you must painstakingly select each individual file and control one at a time.

One of my previous tech tips publicized the fact that you can drag FRM, BAS, CLS, or CTL files from Windows Explorer to the Projects window in VB and VB adds them instantly to the project. What I didn't mention is that you can also drag OCX controls from Explorer and drop them on the VB6 Toolbox to add OCX controls to your project just as quickly and easily.

—**Darin Higgins, Fort Worth, Texas**

VB4/32, VB5, VB6
Level: Intermediate

Add Multicharacter Search Capability to Listboxes

Users often complain that listboxes don't have the same multiple keypress search capabilities as treeviews and other objects. But you can simulate this behavior by adding code to a form with a timer and a listbox whose Sorted property is set to True.

For this test, Form_Load adds some data and sets the default interval between keystrokes. You can type in "AL" to get to Allan instead of the first instance of an entry with an "a" in the list. This can be extremely helpful in long lists. You can also convert this code easily for use within a custom control:

```
Option Explicit

Private Declare Function SendMessage Lib "user32" _
   Alias "SendMessageA" (ByVal hwnd As Long, _
   ByVal wMsg As Long, ByVal wParam As Long, _
   lParam As Any) As Long

Private Const LB_FINDSTRING = &H18F
Private Const LB_ERR = (-1)

Private sSearchstring As String

Private Sub Form_Load()
   With List1
      .AddItem "Adam"
      .AddItem "Allan"
      .AddItem "Arty"
      .AddItem "Aslan"
      .AddItem "Barney"
      .AddItem "Bob"
   End With
   Timer1.Interval = 2000
End Sub

Private Sub List1_KeyPress(KeyAscii As Integer)
   Dim nResult As Long
   Timer1.Enabled = True
   sSearchstring = sSearchstring & Chr$(KeyAscii)
   With List1
      nResult = SendMessage(.hWnd, LB_FINDSTRING, _
         .ListIndex, ByVal sSearchstring)
      If nResult <> LB_ERR Then
         .ListIndex = nResult
         KeyAscii = 0
      End If
   End With
End Sub

Private Sub Timer1_Timer()
   sSearchstring = ""
   Timer1.Enabled = False
End Sub
```

—Joseph L. Scally, Stamford, Conn.

VS.NET
Level: Intermediate

Use Locals to Speed Up Code

When working with an object's fields repetitively in VS.NET, you can improve performance two-fold by storing the object as a local variable rather than a field. In VB.NET, when you use the With myObject ... End With syntax, a local variable is created for myObject. In C#, you must declare the local variable and set it to the object.

—Bill McCarthy, Barongarook, Victoria, Australia

VB4/32, VB5, VB6, SQL Server 7.0
Level: Advanced

Execute a SQL Server DTS Package Remotely

You can easily execute a SQL Server 7.0 Data Transformation Services (DTS) package from VB remotely:

1. Create a DTS package. It can be an import from Excel into SQL Server.
2. Set a reference to Microsoft DTS Package Object Library in any VB project. You might need to load SQL Server on the development machine.
3. Use the LoadFromSQLServer method on the package object:

```
Private Sub cmdRefreshCustomers_Click()
Dim oPackage As New DTS.Package
On Error GoTo eh
'Load the package that we created previously
' ("Customer_List")
'Use the global variables for SQL Server name, UserID,
'and Password.
oPackage.LoadFromSQLServer sServername, sUid, sPwd, _
   DTSSQLStgFlag_Default, _
   "", "", "", "Customer_List", 0
'Execute the Package
oPackage.Execute
MsgBox oPackage.Description, vbInformation, _
   "Re-import Excel sheet."
'Clean up.
Set oPackage = Nothing
Exit Sub
eh:
MsgBox Err.Description, vbCritical, _
   "Error refreshing Customer List"
'For more sophisticated sample VB code with DTS, go
'to the SQL Server 7 CD and browse these folders:
'devtools\samples\dts\dtsempl1 or 2 or 3.
End Sub
```

This is a simple, powerful way to take advantage of any DTS package.

—Steve Simon, Palisades Park, N.J.

VB3, VB4, VB5, VB6
Level: Beginning

Embed Quotation Marks

You use quotation marks in VB to define strings, but how do you include them in your output? Use whichever of these methods works the best for you:

```
   Dim strUseChr As String
   Dim strUseVar As String
   Dim strUseDbl As String

   Const Quote As String = """"

   strUseChr = "Hello " & Chr$(34) & "VB" & _
      Chr$(34) & " World!"
   strUseVar = "Hello " & Quote & "VB" & _
      Quote & " World!"
   strUseDbl = "Hello ""VB"" World!"

   Debug.Print strUseChr
   Debug.Print strUseVar
   Debug.Print strUseDbl
```

Each one prints:

```
   Hello "VB" World!
```

—Dave Keighan, Victoria, British Columbia

VBS
Level: Intermediate

Take a Quick Look at a File

The Windows Script Host (WSH) supports many useful features, including VBScript's FileSystemObject object and the ability to drag and drop filenames. You can drag and drop a data file's icon onto the script (or a shortcut to the script) to see the first 10 lines of a file, or you can click on it to get an input box. You can specify any range of lines if you use arguments in the input box. The code gets the requested lines, puts them in a temporary file, and opens the temp file in Notepad. This utility can come in handy when you want to take a quick look at the layout of lines in a large file.

You can download the WSH from Microsoft's Web site at http://msdn.microsoft.com/scripting. Be sure to download the latest release version if the shortcut doesn't activate with drag-and-drop. Save this code into a file with a VBS extension, create a shortcut on your desktop, then take a quick look at the files:

```
Dim sInputLine, sMain, s
Dim i, iP, iEndFileName
Dim fso, tf, f
Dim nStartPos, iLineCnt
Dim iPopupDelay
Dim varAr

' Edit for your system!
Const TempFile = "C:\Temp\temp.txt"

nStartPos = 1      ' Default first line.
iLineCnt = 10      ' Default number of lines to show.
iPopupDelay = 4    ' Default Popup display, in seconds.

Set objArgs = WScript.Arguments

' If drag and drop was used,
' the argument will be the filename.
If objArgs.Count > 0 Then
   sInputLine = objArgs(0)
Else
   sInputLine = InputBox( _
   "Enter full name of file:" & vbCrLf & vbCrLf _
   & "Arguments allowed after the file name:" & _
   vbCrLf & "     [number of lines to" & _
   "show] [line to start at]" & vbCrLf & _
      "Use single space for argument separator.", _
      "Display Ten Lines of a File", "C:\")
   sInputLine = Trim(sInputLine)
End If

' Clean up as we go.
Set objArgs = Nothing

' If the cancel button was clicked, exit.
If sInputLine = "" Then
   DisplayMsg "No file name entered."
   WScript.quit (0)
End If

' Get start of extension for parsing
' reference point.
i = InstrRev(sInputLine, ".")

' If no extension, exit gracefully.
If i = 0 Then
   DisplayMsg "The filename " & sInputLine & _
   " has no extension."
   WScript.quit (0)
End If

' Check to see If there are arguments at End of
' sInputLine
i = InStr(i, sInputLine, " ")
' first arg = iLineCnt
' second arg = nStartPos (optional)
If i > 0 Then
   iEndFileName = i - 1
   s = Trim(Mid(sInputLine, i))
   If Len(s) > 0 Then
      varAr = Split(s, " ")
      If UBound(varAr) > 0 Then nStartPos = _
         CLng(varAr(1))
      iLineCnt = CInt(varAr(0))
      s = ""
   End If
   sInputLine = Left(sInputLine, iEndFileName)
End If

' Use the scripting file system object to retrieve
' file lines.
Set fso = WScript.CreateObject( _
   "Scripting.FileSystemObject")

' If the file doesn't exist, exit.
If Not (fso.FileExists(sInputLine)) Then
   DisplayMsg "The file " & sInputLine & _
      " does not exit."
   Set fso = Nothing
   WScript.quit (0)
End If

Set tf = fso.OpenTextFile(sInputLine)

' Read iLineCnt file lines starting with line
' nStartPos
i = 1: iP = 0
Do While tf.AtEndOfStream <> True
   sMain = tf.ReadLine
   If i >= nStartPos Then
      s = s & sMain & vbCrLf
      iP = iP + 1
   End If
   i = i + 1
   If iP >= iLineCnt Then Exit Do
Loop

tf.Close

' Save file lines string to a temporary file.
Set f = fso.CreateTextFile(TempFile)
f.Write (s)
f.Close

' Use the script host shell method to open the
' temporary file in editor.
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "notepad " & TempFile

Set fso = Nothing
Set WshShell = Nothing

Sub DisplayMsg(sMsg)
   Set WshShell = _
      WScript.CreateObject("Wscript.Shell")
   WshShell.Popup sMsg, iPopupDelay, _
      "Exiting Windows Script Host", _
      vbOKOnly + vbInformation
   Set WshShell = Nothing
End Sub
```

—Steve Worley, Bainbridge Island, Wash.

VB.NET
Level: Intermediate

Arrays With Non-Zero Lower Bounds

In VB.NET, you can use the System.Array class to create an array with non-zero lower bounds. To do this, use the System.Array.CreateInstance method: Array.CreateInstance(Type, Lengths(), LowerBounds( )).

If the array you create has two or more dimensions, you can cast it to a normal array. This example creates an array of strings equivalent to *Dim sArray(5 To 14, 8 To 27)*:

```
Dim Lengths() As Int32 = {10, 20}
Dim LowerBounds() As Int32 = {5, 8}

Dim myArray As Array = _
   Array.CreateInstance(GetType(String), _
   Lengths, LowerBounds)

' have to declare the array with the correct
' number of dimensions
Dim sArray(,) As String = CType(myArray, _
   String(,))

Dim i As Int32
For i = 0 To sArray.Rank - 1
   Console.WriteLine _
   ("dimension {0} , LowerBound = {1}, _
      UpperBound = {2}", _
   i, sArray.GetLowerBound(i), _
   sArray.GetUpperBound(i))

Next
```

Note: You cannot cast to single dimension arrays because VB.NET creates them as vectors.

—Bill McCarthy, Barongarook, Victoria, Australia

VB3, VB4, VB5, VB6
Level: Beginning

Copy Filenames to a Clipboard

File hierarchies are becoming more complex and file paths longer as the capacity of hard drives increases. There are still many occasions, however, when you can't browse to identify a file to be used—for example, when entering a constant in VB source. Normally, there is no alternative but to type in the path—this can be both tiresome and error-prone.

The solution: Create a new Standard EXE project, delete the default form (Form1), and add a module (Module1 by default). Type this code into Module1:

```
Sub Main()
   Clipboard.SetText Command$
End Sub
```

Under Project | Project Properties, set the Startup Object to Sub Main. Give the project a suitable name (for example, filename) and create the executable (in this case, filename.exe). Create a shortcut to the executable on your desktop. When the name of file is required, browse using Explorer or My Computer. Drag the file and drop it onto the filename shortcut. The full path of the file is now on the clipboard and you can paste it as necessary.

This works because dropping an object onto a program or its shortcut starts it with the name of the object in the command line. The program merely reads the command line and puts it onto the clipboard.

—M.J. Roycroft, Caversfield, Bicester, Oxfordshire, England

VBS
Level: Beginning

Format Strings in Proper Case

VBScript does not support the StrConv() function, which is useful to format strings in proper case. Use this algorithm to help you:

```
Public Function StrConv( _
  ByVal psString, ByVal plFormat) 'As String

  Dim lsString   'As String
  Dim laString   'As String
  Dim liCount    'As Integer
  Dim lsWord     'As String
  Const vbProperCase = 3

  lsString = psString

  Select Case plFormat
     Case vbProperCase
        lsString = LCase(lsString)
        laString = Split(lsString)
        For liCount = 0 To UBound(laString)
           lsWord = laString(liCount)
           If Len(Trim(lsWord)) > 0 Then
              lsWord = UCase(Left(lsWord, 1)) & _
                 Right(lsWord, Len(lsWord) - 1)
              laString(liCount) = lsWord
           End If
        Next liCount
        lsString = Join(laString)
     Case Else
  End Select

  StrConv = lsString
End Function
```

The sample call StrConv("the pHilles wiLL PrevaiL", 3) returns the string 'The Phillies Will Prevail'.

You can use the same name for the corresponding Visual Basic function to facilitate easy adoption of the native version should it ever be supported in future releases of VBScript. If desired, you also can add support for the other StrConv formatting options. VBScript doesn't currently support the Mid statement (as opposed to the Mid function) either, or you could rewrite this algorithm more efficiently using that.

—Brian Egras, Philadelphia

VB3, VB4, VB5, VB6
Level: Beginning

Add Nonkeyboard Characters to Your Project

When creating a project or Web page, you sometimes need to use characters not included on your keyboard—for example, ®, £, §, ©, $^1/_4$, $^1/_2$, $^3/_4$, and so on. Sure, you can use the Chr$() function and create any single character by ASCII code, but you can accomplish this task in a simpler way.

Hold down the Alt key on a keyboard. Using the numeric keypad, *not* the top row numbers, Type 0, then the three-digit ASCII code of the character. Now release the Alt key. For example, to enter the copyright symbol into a string literal, type Alt-0169.

You'll see the designated character on the screen without any additional coding functionality. Easy, isn't it? All you need to know is the ASCII code of the character you want to use. You can look this up in the MSDN Library under Index: ASCII character set (Character Set 128-255), or—easier still—fire up the Character Map applet that comes with Windows.

Be aware that although most text fonts follow mostly standard character mapping, deviations are common, and all bets are off if you end up with a symbol font.

—Alex Grinberg, Holland, Pa.

For even more tricks and tips go to www.vbpj.com or www.vcdj.com

VB4/32, VB5, VB6
Level: Intermediate

Create Unconventional Windows to Dazzle Users

When designing a portion of an application that must grab users' attention quickly—such as your company's splash screen—you might want to create a nonrectangular window. This code shows you how to create a V-shaped window based on nine points:

```
Private Type POINTAPI
   x As Long
   y As Long
End Type

Private Declare Function SetWindowRgn Lib "user32" _
   (ByVal hWnd As Long, ByVal hRgn As Long, _
   ByVal bRedraw As Boolean) As Long

Private Declare Function CreatePolygonRgn _
   Lib "gdi32" (ByRef lpPoint As POINTAPI, _
   ByVal nCount As Long, _
   ByVal nPolyFillMode As Long) As Long

Private Sub Form_Load()
   Dim lhandle As Long
   Dim lpPoint(0 To 8) As POINTAPI

   lpPoint(0).x = 0
   lpPoint(0).y = 0
   lpPoint(1).x = 20
   lpPoint(1).y = 150
   lpPoint(2).x = 60
   lpPoint(2).y = 150
   lpPoint(3).x = 80
   lpPoint(3).y = 0
   lpPoint(4).x = 52
   lpPoint(4).y = 0
   lpPoint(5).x = 46
   lpPoint(5).y = 120
   lpPoint(6).x = 40
   lpPoint(6).y = 120
   lpPoint(7).x = 32
   lpPoint(7).y = 0
   lpPoint(8).x = 0
   lpPoint(8).y = 0

   lhandle = CreatePolygonRgn(lpPoint(0), 9, 1)
   Call SetWindowRgn(Me.hWnd, lhandle, True)
End Sub
```

—Andrew Holliday, Phoenix

VB3, VB4, VB5, VB6
Level: Beginning

Close All Child Forms in One Shot

In MDI applications, a user might have two or three or even more MDI child windows open at any given time. But in applications where you have user log-in and log-out security, you likely want to unload all open forms when the user logs out. To accomplish this, use this small piece of code:

```
Do Until MDIform1.ActiveForm Is Nothing
   Unload MDIform1.ActiveForm
Loop
```

If you need to save any values in any form by default, you can include a call to the appropriate Save method in the Unload event of that form.

—Unnikrishnan Thampy, Floral Park, N.Y.

VB3, VB4, VB5, VB6
Level: Beginning

Return Roman Numerals

This VB procedure returns decimal numbers (integers) as Roman numerals (a string), ranging from 1 to 4999. Numbers outside this range return the same number as a string. The optional parameter iStyle allows two different numerical styles: standard (4 = iv, 9 = ix, and so on) when iStyle = -1, or classical (4 = iiii, 9 = viiii, and so on) when iStyle = -2.

The variable *x* should make the function more efficient, although you might not notice the time saved on a fast machine:

```
Public Function Roman(ByVal n As Integer, _
   Optional iStyle As Integer = -1) As String

   If n < 1 Or n >= 5000 Then
      Roman = CStr(n)
      Exit Function
   End If

   If iStyle <> -2 Then iStyle = -1

   Dim sRtn As String, i As Integer, x As Integer
   Dim r(1 To 13) As String, v(1 To 13) As Integer

   r(1) = "i": v(1) = 1
   r(2) = "iv": v(2) = 4
   r(3) = "v": v(3) = 5
   r(4) = "ix": v(4) = 9
   r(5) = "x": v(5) = 10
   r(6) = "xl": v(6) = 40
   r(7) = "l": v(7) = 50
   r(8) = "xc": v(8) = 90
   r(9) = "c" v(9) = 100
   r(10) = "cd": v(10) = 400
   r(11) = "d": v(11) = 500
   r(12) = "cm": v(12) = 900
   r(13) = "m": v(13) = 1000

   x = UBound(v)
   sRtn = ""
   Do
      For i = x To LBound(v) Step iStyle
         If v(i) <= n Then
            sRtn = sRtn & r(i)
            n = n - v(i)
            x = i
            Exit For
         End If
      Next i
   Loop Until n = 0

   Roman = sRtn
End Function
```

—Steven Digby, London

VB3, VB4, VB5, VB6
Level: Beginning

Code an Event Procedure for Each Textbox

If you want to code an event procedure (such as GotFocus) for each textbox on a freshly designed form, you must switch manually from the Change event to the GotFocus event for each one. This can be annoying and tedious, especially when many textboxes don't belong to a control array. To get around this, double-click on each textbox on the form to generate an empty Change event procedure. Then do a find-and-replace, searching for "_Change" and replacing it with "_GotFocus". Be careful not to do a "Replace All" unless there's very little other code in that form module already.

—Thomas R. Weiss, Deerfield, Ill.

VB5, VB6
Level: Intermediate

Use Hidden Enum Bounds

Enumerated parameters don't prevent you from passing unenumerated data to the function. Let's say you have this enumeration:

```
Public Enum geAccessType
   ReadOnly = 1
   WriteOnly = 2
   ReadWrite = 3
   NoAccess = 4
End Enum
```

And you have this function:

```
Public Function DoSomeJob( _
   eType As geAccessType) As Long
   MsgBox eType
End Function
```

You can call this function like this:

```
'call #1
DoSomeJob ReadOnly
```

You can also call it like this, passing a value other than the enumerated constants:

```
'call #2
DoSomeJob 45
```

In any case, it works.

If you add two variables to the enumeration and modify your function implementation, you can prevent out-of-bounds cases such as call #2 easily:

```
Public Enum geAccessType
   [_minAccessType] = 1
   ReadOnly = 1
   WriteOnly = 2
   ReadWrite = 3
   NoAccess = 4
   [_maxAccessType] = 4
End Enum

Public Function DoSomeJob( _
   eType As geAccessType) As Long
   Select Case eType
      Case geAccessType.[_minAccessType] To _
         geAccessType.[_maxAccessType]
            MsgBox eType
      Case Else
         'raise error or do something else
   End Select
End Function
```

By default, [_minAccessType] or [_maxAccessType] do not appear in the constant list. If you want to see them, open the Object Browser, right-click inside it, and select Show Hidden Members.

—Russ Kot, Northbrook, Ill.

VB4/32, VB5, VB6
Level: Intermediate

International Test for Illegal Characters

I was interested to read the tip "Test for Illegal Characters" in the 10th Edition of the "101 Tech Tips for VB Developers" supplement [*Visual Basic Programmer's Journal* February 2000]. The tip, however, has two significant drawbacks as published. First, it requires a function from the SHLWAPI DLL, which requires either Win98/2000 or Win95/NT with Internet Explorer 4.0 or higher. Second, it only works, as presented, for U.S. (7-bit) character sets, requiring those of us who work with international character sets (such as accented characters) to consider which characters will be legal where our apps run.

Luckily, Windows has the solution: the IsCharAlphaNumeric function, defined in User32.dll. This function uses the currently defined locale when performing comparisons, thereby allowing full use of accented characters. This sample demonstrates how you might use this function:

```
Public Declare Function IsCharAlphaNumeric Lib _
   "user32" Alias "IsCharAlphaNumericA" ( _
   ByVal cChar As Byte) As Long

Public Function IsAlphaNum(ByVal sInput As String) _
   As Boolean
   Dim fCheck As Boolean
   Dim i As Integer

   ' Assume non-alphanumeric
   fCheck = False

   ' If we don't have any input, drop out
   If Len(sInput) Then
      i = 0
      Do
         i = i + 1
         fCheck = _
            CBool(IsCharAlphaNumeric( _
            Asc(Mid$(sInput, i, 1))))
      Loop While fCheck And (i < Len(sInput))
   End If
   IsAlphaNum = fCheck
End Function
```

You may pass any single or multiple character string to the function IsAlphaNum. The return value will be True if *all* characters are alphanumeric and False otherwise.

Windows also has several other useful functions for working with characters in the current locale. Note, however, that all functions require a byte to be passed, which you can achieve by passing the Asc() value of a given character (see previous example):

```
' Check if a given character is alphabetic
Public Declare Function IsCharAlpha Lib "user32" _
   Alias "IsCharAlphaA" (ByVal cChar As Byte) _
   As Long

' Check if a given character is lowercase
Public Declare Function IsCharLower Lib "user32" _
   Alias "IsCharLowerA" (ByVal cChar As Byte) _
   As Long

' Check if a given character is uppercase
Public Declare Function IsCharUpper Lib "user32" _
   Alias "IsCharUpperA" (ByVal cChar As Byte) _
   As Long
```

—John Cullen, Pedroucos, Portugal

For even more tricks and tips go to
www.vbpj.com or www.vcdj.com

VB5, VB6
Level: Intermediate

Use CopyFromRecordset With ODBC Recordsets

You can create an ODBCDirect recordset for use with the Excel Range object's CopyFromRecordset method by using the DAO.Connection object's OpenRecordset method:

```
Public Function CreateDaoRecordset( _
   ByVal sDataSource As String, _
   ByVal sUser As String, _
   ByVal sPwd As String, _
   ByVal sSql As String) _
   As DAO.Recordset

   Dim daoWs As Workspace
   Dim daoConn As DAO.Connection
   Dim sConn As String
   Dim dbEng As DBEngine
   Set dbEng = New DBEngine
   Set daoWs = dbEng.CreateWorkspace("", "admin", _
      "", dbUseODBC)
   sConn = "ODBC;DSN=" & sDataSource & ";UID=" _
      & sUser & ";PWD=" & sPwd
   Set daoConn = daoWs.OpenConnection("", , , sConn)
   Set CreateDaoRecordset = _
      daoConn.OpenRecordset(sSql, dbOpenSnapshot)
End Function
```

The CopyFromRecordset method also works with Oracle8 databases. The trick: You must use a proper ODBC driver. The Microsoft ODBC driver for Oracle, msorcl32.dll version 02.573.3513.0, doesn't support the NUMBER data type in this method. The Oracle ODBC driver, sqora32.dll version 8.0.5.0.0, treats the NUMBER(n) data type as a dbDecimal and generates "Unspecified Automation Error" in the CopyFromRecordset method. But it accepts the NUMBER data type (without precision), interpreting it as a dbDouble.

CopyFromRecordset doesn't copy column names to the Excel worksheet for further data analysis or reporting, so use this simple code instead. It copies column names to the first row of the active Excel worksheet oWsh and copies all data from the daoRs Recordset. The code assumes oWsh and daoRs have been declared and initialized elsewhere:

```
oWsh.Activate
          For iCol = 0 To daoRs.Fields.Count - 1
             oWsh.Cells(1, iCol + 1).Value = _
                daoRs.Fields(iCol).Name
          Next
          oWsh.Range("A2").CopyFromRecordset daoRs
```

—Leonid Strakovskiy, New York

SQL Server 6.5 and up
Level: Beginning

Simple Way to Debug a Stored Procedure

Debugging stored procedures can be a headache, but here's an easier way to trace a stored procedure's execution: Use the PRINT statement. PRINT lets you output and analyze variable values, which is sometimes good enough. Note a few restrictions when using PRINT:

1. You can use only Char or VarChar data types. You must convert other data types to Char or VarChar in order to "print" them out.
2. The printed string can be up to 8,000 characters long; any characters after 8,000 are truncated. (SQL Server 6.5 up to 255 characters long.)
3. SQL Sever 6.5 doesn't allow inline concatenation of variables. SQL Server 7.0 and 2000 don't have this limitation.

—Alex Grinberg, Holland, Pa.

VB3, VB4, VB5, VB6
Level: Intermediate

Format Color for HTML

This function, which converts a color value into a string suitable for HTML, formats an RGB color value, palette index, or system color constant. You accomplish this by breaking out the individual color values for red, green, and blue, then recombining them in the opposite order Windows likes, so HTML renderers will provide the correct color. The call to OleTranslateColor ensures you're using an actual color reference, by dereferencing system color constants or palette indices:

```
Public Function HtmlHexColor(ByVal ColorValue As _
   Long) As String
   Dim r As Byte
   Dim g As Byte
   Dim b As Byte

   ' convert color if needed
   Call OleTranslateColor( _
      ColorValue, 0&, ColorValue)

   ' break out color bytes
   r = (ColorValue Mod &H100)
   g = (ColorValue \ &H100) Mod &H100
   b = (ColorValue \ &H10000) Mod &H10000

   ' format the return string
   HtmlHexColor = "#" & _
      Right$("0" & Hex$(r), 2) & _
      Right$("0" & Hex$(g), 2) & _
      Right$("0" & Hex$(b), 2)
End Function
```

—Monte Hansen, Ripon, Calif.

VBS
Level: Beginning

Use the Immediate If Function in VBScript

Visual Basic includes the Immediate If function (IIf), but VBScript (VBS) does not. However, you can copy this code to VBS to allow the IIf function to be used:

```
'VB Function not included in VBS
Function IIf(Expression, TruePart, FalsePart)
   If Expression = True Then
      If IsObject(TruePart) Then
         Set IIf = TruePart
      Else
         IIf = TruePart
      End If
   Else
      If IsObject(FalsePart) Then
         Set IIf = FalsePart
      Else
         IIf = FalsePart
      End If
   End If
End Function
```

The function can return both objects and basic data types. Here's a sample function from an ASP page that calls the IIf function:

```
' Return a True or False value for a checkbox
Function CheckBoxValue(Name)
   CheckBoxValue = _
      IIf(Request.Form(Name) = "on", True, False)
End Function
```

—Conrad Sollitt, Los Angeles

VB4/32, VB5, VB6
Level: Advanced

☆☆☆☆☆ Five Star Tip

Override Built-In Keywords

You can override some of the built-in VB keywords with your own version of the function. For instance, FileDateTime is a handy built-in function in VB, but it suffers from one big problem: It can't set the date/time of a file. By overriding the built-in function, however, you can provide this feature. With this approach, the function can determine for itself how it is being used and perform accordingly.

You can override a number of keywords and functions in this manner:

```
Private Declare Function SystemTimeToFileTime Lib _
   "kernel32" (lpSystemTime As SYSTEMTIME, _
   lpFileTime As FILETIME) As Long
Private Declare Function LocalFileTimeToFileTime _
   Lib "kernel32" (lpLocalFileTime As FILETIME, _
   lpFileTime As FILETIME) As Long
Private Declare Function CreateFile Lib "kernel32" _
   Alias "CreateFileA" (ByVal lpFileName As _
   String, ByVal dwDesiredAccess As Long, ByVal _
   dwShareMode As Long, lpSecurityAttributes As _
   Any, ByVal dwCreationDisposition As Long, _
   ByVal dwFlagsAndAttributes As Long, _
   ByVal hTemplateFile As Long) As Long
Private Declare Function SetFileTime Lib "kernel32" _
   (ByVal hFile As Long, lpCreationTime As Any, _
   lpLastAccessTime As Any, lpLastWriteTime As _
   Any) As Long
Private Declare Function CloseHandle Lib "kernel32" _
   (ByVal hObject As Long) As Long

Private Type FILETIME
   dwLowDateTime As Long
   dwHighDateTime As Long
End Type

Private Type SYSTEMTIME
   wYear As Integer
   wMonth As Integer
   wDayOfWeek As Integer
   wDay As Integer
   wHour As Integer
   wMinute As Integer
   wSecond As Integer
   wMilliseconds As Integer
End Type

Private Const GENERIC_WRITE As Long = &H40000000
Private Const FILE_SHARE_READ As Long = &H1
Private Const FILE_SHARE_WRITE As Long = &H2
Private Const OPEN_EXISTING As Long = 3

Public Function FileDateTime(ByVal FileName As String, _
   Optional ByVal TimeStamp As Variant) As Date

   ' Raises an error if one occurs just like FileDateTime

   Dim x As Long
   Dim Handle As Long
   Dim System_Time As SYSTEMTIME
   Dim File_Time As FILETIME
   Dim Local_Time As FILETIME

   If IsMissing(TimeStamp) Then
      'It's missing so they must want to GET the timestamp
      'This acts EXACTLY like the original built-in function
      FileDateTime = VBA.FileDateTime(FileName)
   ElseIf VarType(TimeStamp) <> vbDate Then
      'You must pass in a date to be valid
      Err.Raise 450
   Else
      System_Time.wYear = Year(TimeStamp)
      System_Time.wMonth = Month(TimeStamp)
      System_Time.wDay = Day(TimeStamp)
      System_Time.wDayOfWeek = _
         Weekday(TimeStamp) - 1
      System_Time.wHour = Hour(TimeStamp)
      System_Time.wMinute = Minute(TimeStamp)
      System_Time.wSecond = Second(TimeStamp)
      System_Time.wMilliseconds = 0

      'Convert the system time to a file time
      x = SystemTimeToFileTime(System_Time, Local_Time)

      'Convert local file time to file time based on UTC
      x = LocalFileTimeToFileTime(Local_Time, File_Time)

      'Open the file so we can get a file handle to
      'the file
      Handle = CreateFile(FileName, GENERIC_WRITE, _
         FILE_SHARE_READ Or FILE_SHARE_WRITE, _
         ByVal 0&, OPEN_EXISTING, 0, 0)
      If Handle = 0 Then
         Err.Raise 53, "FileDateTime", _
            "Can't open the file"
      Else
         'Now change the file time and date stamp
         x = SetFileTime(Handle, ByVal 0&, _
            ByVal 0&, File_Time)
         If x = 0 Then
            'Error occured
            Err.Raise 1, "FileDateTime", _
               "Unable to set file timestamp"
         End If
         Call CloseHandle(Handle)
         'Return newly set date/time
         FileDateTime = VBA.FileDateTime(FileName)
      End If
   End If
End Function
```

—Darin Higgins, Fort Worth, Texas

VB4/32, VB5, VB6, SQL Server 6.5 and up, Oracle 8*i* and up
Level: Beginning

Compare Oracle and SQL Server Dates

Oracle and SQL Server databases use different date/time resolutions, which poses a problem when you compare times from the two databases: The times will rarely be equal. Solve this problem by allowing for a margin of error. Treat the dates and times as floating-point numbers and remember that each day is equal to the whole number 1, and there are 86,400 seconds in a day. This function matches times within five seconds (default) of one another:

```
Public Function MatchTime(adoFldOracle As ADODB.Field, _
   adoFldSQLServer As ADODB.Field, _
   Optional ByVal Tolerance As Long = 5) As Boolean

   Dim dtOracle As Date
   Dim dtSQLServer As Date
   Dim dblTolerance As Double
   Const OneSecond As Double = 1 / 86400

   dblTolerance = OneSecond * Tolerance
   dtOracle = adoFldOracle.Value
   dtSQLServer = adoFldSQLServer.Value
   If ((dtOracle > (dtSQLServer + dblTolerance)) Or _
      (dtOracle < (dtSQLServer - dblTolerance))) Then
      MatchTime = False
   Else
      MatchTime = True
   End If
End Function
```

—Andy Clark, Richmond, Va.

VB3, VB4, VB5, VB6
Level: Beginning

Manage Errors With Sparse Line Numbering

You might have used line numbering to track error locations, but this technique can be a pain (and ugly) to use for every line. Sparse line numbers help you locate code sections generating errors through the intrinsic Erl (error line) property.

Erl captures the most recent line number so you can pinpoint error locations with whichever precision you desire. This can help you determine what to do in the error handler (download this code). For example, do you need to roll back the database transaction?

—Bob Hiltner, Seattle

VB3, VB4, VB5, VB6
Level: Intermediate

Replace All Occurrences of One String With Another String

All programmers—especially database programmers—require a function that replaces all occurrences of one substring with another string. For example, they need to replace the single quotes in strings passed to an Oracle database with two single quotes.

Using recursion in this algorithm limits its usefulness slightly below that of VB6's native Replace function, as the time required increases greatly in relation to the length of the searched string:

```
Public Function strReplace(ByVal strString As _
    String, ByVal strToBeReplaced As String, ByVal _
    strReplacement As String, Optional ByVal _
    intStartPosition As Integer = 1) As String
    Dim strNewString As String
    Dim intPosition As Integer

    On Error GoTo ErrorHandler

    ' intStartPosition will be one initially
    intPosition = InStr(intStartPosition, _
        strString, strToBeReplaced)

    If intPosition = 0 Then
        ' Nothing more to do so return final string
        strReplace = strString
    Else
        strNewString = Left$(strString, intPosition _
            - 1) & strReplacement & Mid$(strString, _
            intPosition + Len(strToBeReplaced))
        ' Recursively call strReplace until there are
        ' no more occurrences of the string to be
        ' replaced in the string passed in. We now only want
        ' to process the remaining unprocessed part of the
        ' string so we pass a start position.
        strReplace = strReplace(strNewString, _
            strToBeReplaced, strReplacement, _
            intPosition + Len(strReplacement))
    End If
Exit Function

ErrorHandler:
    ' Place error handler code here
End Function
```

—Patrick Tighe, Eastwall, Dublin, Ireland

VB6
Level: Advanced

☆☆☆☆☆ **Five Star Tip**

Serialize Data Using a PropertyBag

You can serialize your data quickly by placing it into a PropertyBag object, then reading the PropertyBag's Contents property. This property is really a Byte array that is a serial representation of the data in your PropertyBag object. You can use this byte array for many purposes, including an efficient means of data transmission over DCOM:

```
Private Function PackData() As String
    Dim pbTemp  As PropertyBag

    'Create a new PropertyBag object
    Set pbTemp = New PropertyBag
    With pbTemp
        'Add your data to the PB giving each item a
        'unique string key
        Call .WriteProperty("FirstName", "John")
        Call .WriteProperty("MiddleInitial", "J")
        Call .WriteProperty("LastName", "Doe")

        'Place the serialized data into a string
        'variable.
        Let PackData = .Contents
    End With

    Set pbTemp = Nothing
End Function
```

To retrieve the serialized data, simply create a new PropertyBag object and set the serialized string to its Contents property. Convert the string into a byte array before assigning it to the Contents property:

```
Private Sub UnPackData(sData As String)
    Dim pbTemp  As PropertyBag
    Dim arData()   As Byte

    'Convert the string representation of the data to
    'a Byte array
    Let arData() = sData

    'Create a new PropertyBag object
    Set pbTemp = New PropertyBag
    With pbTemp
        'Load the PropertyBag with data
        Let .Contents = arData()

        'Retrieve your data using the unique key
        Let m_sFirstName = .ReadProperty("FirstName")
        Let m_sMiddleInitial = _
            .ReadProperty("MiddleInitial")
        Let m_sLastName = .ReadProperty("LastName")
    End With

    Set pbTemp = Nothing
End Sub
```

—Mike Kurtz, McKees Rocks, Pa.

VS.NET
Level: Beginning

Clear a Picture Property at Design Time

To clear the picture property of a control at design time, right-click on the icon next to the entry in the Properties window and select Reset from the popup menu.

—Bill McCarthy, Barongarook, Victoria, Australia

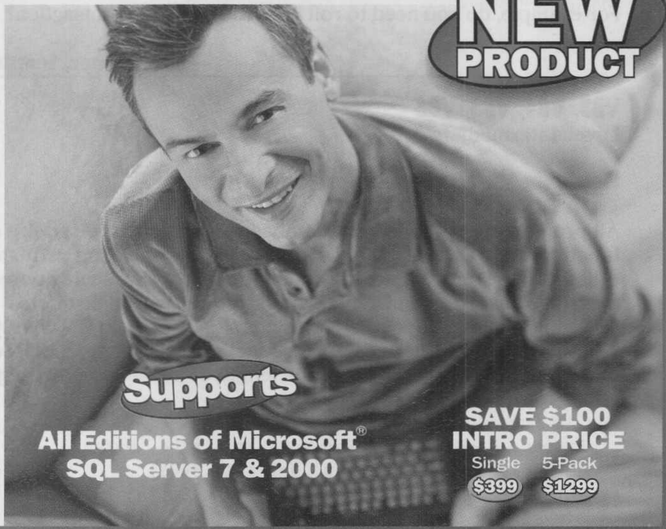
Take Control of SQL Server

FREE Catalog & Demo
1-877-367-4032

Total SQL Analyzer™

Total SQL Analyzer is the world's first professional documentation and analysis program for Microsoft® SQL Server. Now you finally can take control of SQL Server.

- Get complete documentation of your server configuration.
- Get in-depth analysis of potential issues and performance problems.
- Use our powerful documentation explorer to see new views of your server and databases.
- Get complete documentation of database objects, including all properties and settings.
- Use over 70 presentation-quality reports. Print, Preview, or export to Word or HTML
- Easily see TSQL across all your stored procedures.



NEW PRODUCT

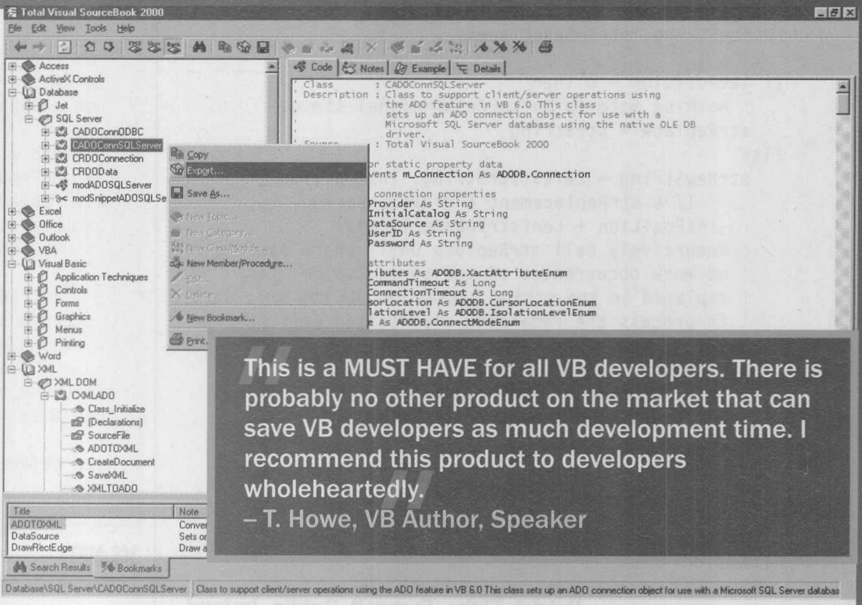
Supports All Editions of Microsoft® SQL Server 7 & 2000

SAVE $100 INTRO PRICE
Single $399 5-Pack $1299

Royalty-Free Source Code, Guaranteed to Improve Productivity!

Total Visual SourceBook™ 2000

"SourceBook saves me hour after hour when I develop. The XML bits have been great for me as I'm fairly new XML, but also the more 'general' code bits give me what I need. And if I don't find exactly what I need, I can always use one or more of the code samples as a template for creating what I want! Thanks!"
– C. Thomsen, Microsoft MVP VB



```
' Class       : CADOConnSQLServer
' Description : Class to support client/server operations using
'               the ADO feature in VB 6.0 This class
'               sets up an ADO connection object for use with a
'               Microsoft SQL Server database using the native OLE DB
'               driver.
' Source      : Total Visual SourceBook 2000

r static property data
vents m_Connection As ADODB.Connection

 connection properties
Provider As String
InitialCatalog As String
DataSource As String
UserID As String
Password As String

attributes
ributes As ADODB.XactAttributeEnum
CommandTimeout As Long
ConnectionTimeout As Long
sorLocation As ADODB.CursorLocationEnum
lationLevel As ADODB.IsolationLevelEnum
e As ADODB.ConnectModeEnum
```

Title	Note
ADOTOXML	Conver
DataSource	Sets or
DrawRectEdge	Draw a

Database\SQL Server\CADOConnSQLServer | Class to support client/server operations using the ADO feature in VB 6.0 This class sets up an ADO connection object for use with a Microsoft SQL Server databas

"This is a MUST HAVE for all VB developers. There is probably no other product on the market that can save VB developers as much development time. I recommend this product to developers wholeheartedly."
– T. Howe, VB Author, Speaker

"Total Visual SourceBook 2000 is a valuable tool that I refer to frequently. It is not only an easy to use reference for reliable code, it also uses good coding practices that have improved the quality of my own procedures and applications."
– N. Chezem, VB Developer

"I really love this program! Not only is it a time-saver and easy to use, but to be able to organize your own code is such an added bonus."
– Connie P., VB Developer

Single $399 5-Pack $1299

Visual Basic Programmer's Journal '99 Readers' Choice Merit · WinPlanet ★★★★½ 4½ out of 5 stars! · VB 2000 Tips & Tricks Reader Choice Award · VBPJ 2001 Readers' Choice Award

1-877-367-4032 · fmsinc.com/tt · sales@fmsinc.com

FMS
Since 1986